Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

# Creación de una aplicación de apoyo a los cambios normativos en protección de datos

Trabajo Fin de Grado

**Grado en Ingeniería Informática**

**Autor**: Álvaro Morro Ibáñez

**Tutor**: Juan Vicente Oltra Gutiérrez

[2016-2017]

# Resumen

Tras la aprobación del Reglamento General de Protección de datos, las empresas se enfrentan a la tarea de adaptarse a esta nueva normativa. No hacerlo, puede acarrear severas sanciones que pueden llegar a los veinte millones de euros. Las empresas necesitan conocer en qué grado cumplen con la nueva legislación y qué acciones necesitan llevar a cabo para adaptarse a ella.

Para apoyar este cometido, se ha desarrollado una aplicación que, mediante la realización de un cuestionario, permite a las empresas saber en qué áreas deben mejorar y cuáles son sus mayores deficiencias en materia de protección de datos. La aplicación genera un informe en el cual se listan todos los problemas encontrados para facilitar la corrección de los mismos. Este informe puede ser empleado por un analista para recomendar las acciones que necesita llevar a cabo la empresa para adaptarse correctamente al nuevo reglamento europeo de protección de datos.

**Palabras clave:** Aplicación, Protección de datos, Cuestionario, Informe.

# Abstract

Following the adoption of the General Data Protection Regulation, companies are faced with the task of adapting to this new regulation, if it is not done they face severe sanctions that can reach up to twenty million euros. Companies need to know the degree to which they comply with the new legislation and what actions they need to take to fulfil it.

To support this task, an application has been developed which, by means of a questionnaire, allows companies to know in which areas they should improve and which are their biggest deficiencies in data protection. The application generates a report that lists all the problems found to facilitate the correction of the same. This report may be used by an analyst to recommend the actions necessary to be carried out by the company to properly adapt to the new european data protection regulation.

**Keywords:** Aplicación, Data protecction, Questionnaire, Report.

# Tabla de contenidos

# 1. Objeto y objetivos

El objeto del presente Trabajo de Fin de Grado es la obtención del título de Graduado en Ingeniería Informática expedido por la Universidad Politécnica de Valencia.

El objetivo de este Trabajo de Fin de Grado es el diseño y desarrollo de una aplicación que facilite a las empresas la creación de un informe sobre sus procedimientos de tratamiento de datos personales para poder identificar si se adaptan o no al nuevo reglamento europeo de protección de datos.

Esta aplicación debe ser fácil de mantener y mejorar, además debe permitir a los usuarios utilizarla de forma sencilla e intuitiva, aportándoles una experiencia de uso satisfactoria.

# 2. Introducción

Con la aprobación del nuevo Reglamento General de Protección de Datos de la unión europea, las empresas y organizaciones se enfrentan a la labor de adaptar sus procedimientos a este nuevo marco jurídico.

El Reglamento General de Protección de Datos (RGPD) persigue implantar una legislación uniforme a nivel europeo para proporcionar a las empresas seguridad jurídica al mismo tiempo que protege y amplia los derechos de los consumidores introduciendo nuevos derechos, así como ampliando otros ya existentes.

El nuevo reglamento afecta de manera notoria a las empresas, las cuales se ven en la obligación de adaptar sus procedimientos a la hora de obtener, almacenar y tratar datos de carácter personal. Esta tarea puede ser de enorme complejidad debido a los cada vez más altos volúmenes de datos que manejan lo que hace necesario disponer de herramientas que faciliten a las empresas el análisis de sus procedimientos.

Para facilitar el diagnóstico de sus procesos y actividades se ha desarrollado una aplicación usando el lenguaje de programación Java, la cual permite realizar informes mediante la realización de un cuestionario. Dichos informes pueden ser posteriormente exportados a diferentes formatos facilitando de esta manera la labor del analista. De esta manera se consigue un informe sobre la situación actual de la empresa en cuanto al cumplimiento del Reglamento General de Protección de Datos, permitiéndole actuar realizando las modificaciones oportunas para adaptarse al nuevo reglamento.

Esta aplicación está pensada para aquellas empresas que no disponen de los recursos necesarios (tanto económicos como humanos) para llevar a cabo la adaptación al RGPD, la aplicación pretende ayudar en las labores de análisis, pero seguirá siendo necesaria la intervención de un profesional que determine las acciones a llevar a cabo a partir del informe generado por la aplicación.

Actualmente, no existe en el mercado ninguna aplicación como la que se ha planteado para este trabajo. Existe una gran cantidad de bibliografía sobre el nuevo reglamento, sus cambios más importantes, así como guías que pretenden facilitar a las empresas el proceso de adaptación a este nuevo reglamento.

## 2.1 Estructura del trabajo

La presente memoria está estructurada en ocho apartados. A continuación, se explica el contenido de cada uno de ellos.

El primer apartado de la memoria es el de Objeto y objetivos, en él se explica cuál es el objeto del trabajo, así como los objetivos que se persiguen con el mismo.

En el segundo apartado, Introducción, se exponen las razones por las que se ha desarrollado este proyecto destacando la complejidad del nuevo reglamento europeo de protección de datos y como este afecta a las empresas.

En el apartado de desarrollo, se explica cómo se ha desarrollado la aplicación, comenzando por la metodología empleada, las tecnologías utilizadas y el diseño del cuestionario para, a continuación, detallar la arquitectura desarrollada. Se explican las clases más importantes que forman la aplicación, así como la implementación de la persistencia, la generación de informes y el apartado gráfico del software. Por último, en este apartado se explica cómo se han diseñado las pruebas unitarias necesarias para asegurar un buen funcionamiento de la aplicación.

El apartado Guía de uso, sirve de guía para los usuarios que utiliza por primera vez la aplicación explicándose como utilizarla correctamente, así como explicando las posibilidades que tiene.

Por último, se hallan las conclusiones del trabajo, el anexo, el glosario y la bibliografía.

## 2.2 Sobre el reglamento europeo de protección de datos

El veinticinco de mayo de dos mil dieciséis entró en vigor el Reglamento General de Protección de Datos, aunque no será de obligado cumplimiento hasta el veinticinco de mayo de dos mil dieciocho. Este periodo de dos años está destinado a permitir que las empresas realicen los cambios necesarios para adaptarse al nuevo reglamento.

“Esta nueva regulación, que por primera vez se hace a través de un reglamento europeo, comportará cambios significativos en la protección de datos de carácter personal, tanto desde el punto de vista de los derechos de las personas, como de las obligaciones de las personas y entidades que tratan datos de carácter personal.” (Autoridad Catalana de Protección de Datos, 2017 [11])

Este nuevo reglamento pretende unificar la legislación en materia de protección de datos de los estados miembros y crear de esta manera un marco jurídico único para todos los países de la unión que de seguridad a las empresas que operan en la misma. Por ello, este nuevo reglamento introduce una gran cantidad de cambios respecto a la directiva europea a la que sustituye.

Que el periodo destinado a la adaptación de las empresas sea de dos años es una muestra de la complejidad de este nuevo reglamento, así como de la profundidad de sus cambios algunos de los cuales son de enorme calado, pudiéndose hablar de un "nuevo modelo europeo de protección de datos" (Hay Derecho, 2017 [3]).

Los cambios más importantes que introduce el RGPD son:

- Los nuevos principios de responsabilidad y transparencia.

- Nuevas obligaciones para las empresas y organizaciones entre las que destacan: la obligatoriedad, para determinadas empresas, de disponer de un delegado de protección de datos, la creación de una ventanilla única, la obligatoriedad de notificar las brechas de seguridad, la ampliación de los datos que se consideran sensibles, etc.

- Nuevos derechos para los ciudadanos entre los que destacan: el derecho al olvido, el derecho a exigir indemnizaciones por daños y perjuicios derivados del tratamiento ilícito de datos personales, el derecho a solicitar la transferencia de los datos personales de un proveedor de servicios en internet a otro, el derecho a solicitar el bloqueo temporal del tratamiento de sus datos cuando existan dudas sobre su licitud, etc.

Como se ha podido comprobar, las empresas y organismos deberán introducir muchos cambios en su forma de actuar, adoptando medidas en materia de prevención, de forma que puedan demostrar que están condiciones de cumplir las normas que establece el nuevo reglamento ya que, en caso de no cumplirlas, se contemplan unas estrictas sanciones de hasta veinte millones de euros o del cuatro por ciento de la facturación anual lo cual hace fundamental para las empresas el asegurarse de que cumplen la normativa de forma correcta.

Sin embargo, las empresas, sobre todo las más pequeñas y que, por lo tanto, disponen de menos recursos, tienen problemas para adaptarse a la nueva normativa. Según un reciente estudio llevado a cabo por la empresa tecnológica NetApp (NetApp, 2017 [4]), más del setenta por ciento de los responsables y directores tecnológicos de empresas europeas se muestran preocupados, en mayor o menor medida, ante la posibilidad de que sus organizaciones no logren adecuarse al nuevo Reglamento General de Protección de Datos dentro de los plazos estipulados por los organismos europeos.

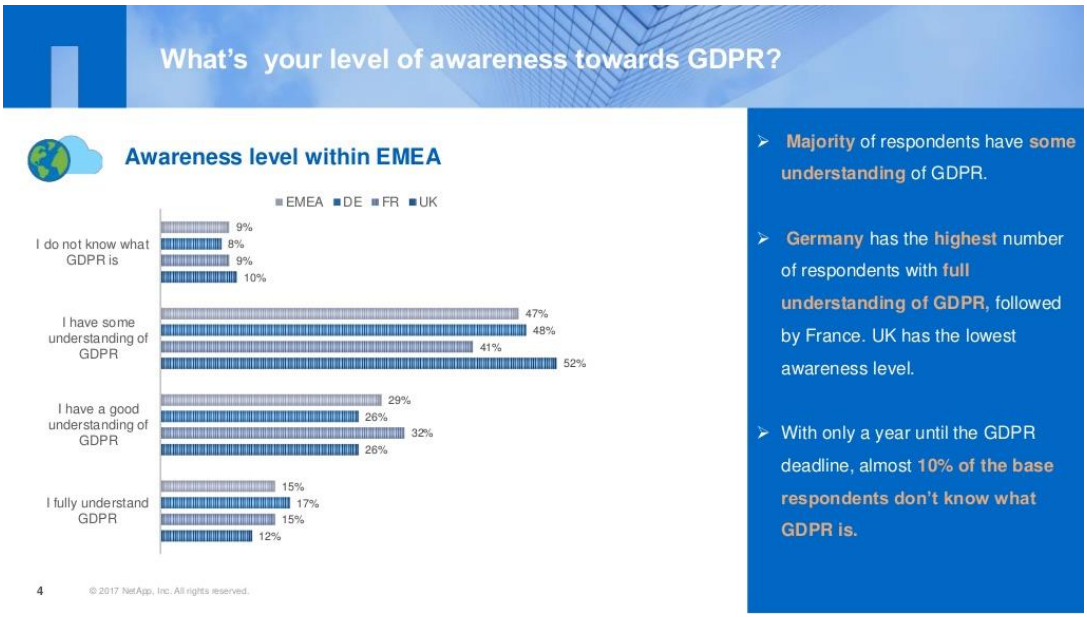


*Ilustración 1: Entendimiento del nuevo reglamento europeo de protección de datos. NetAPP, https://www.slideshare.net/NetAppUK/netapp-cloud-gdpr-response-survey-emea-findings-ii*

Además, solo un treinta y cinco por ciento de los encuestados reconoce haber completado todos los preparativos necesarios para adaptarse al nuevo reglamento y cerca del quince por ciento dice no haber tomado aún ninguna medida en su empresa.

Como se puede observar, la adaptación de las empresas y organizaciones a esta nueva legislación no es una tarea trivial y se requiere de especialistas en la materia tanto para analizar el funcionamiento de las empresas como para dictaminar las acciones necesarias para llevar a cabo la adaptación a la nueva normativa.

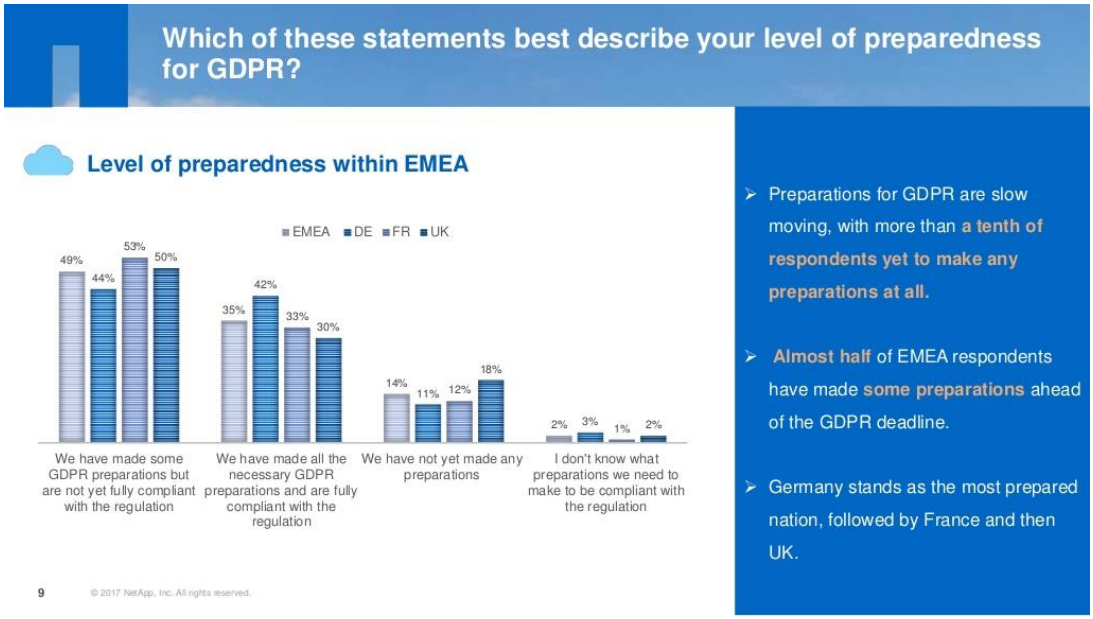


*Ilustración 2: Nivel de preparación para el nuevo reglamento. NetAPP, https://www.slideshare.net/NetAppUK/netapp-cloud-gdpr-response-survey-emea-findings-ii*

# 3. Desarrollo

La aplicación ha sido desarrollada en el lenguaje Java ya que se trata de un lenguaje de programación ampliamente utilizado en la industria y que se ha estudiado durante el grado de ingeniería informática. Como entorno de desarrollo se ha utilizado Eclipse al tratarse de una herramienta gratuita y también ha sido empleada en el grado. Para el apartado gráfico de la aplicación se ha empleado la librería [2] *JavaFX*, mientras que para la generación de los informes se ha empleado la librería *Dynamic Reportos* por tratarse de una herramienta gratuita, escrita en Java y basada en *Jasper Reports* uno de los generadores de informes más populares. Además, se ha empleado software de control de versiones como GitHub y de almacenamiento en la nube como Google Drive durante el desarrollo de la misma.

## 3.1 Análisis de los requisitos

Los requisitos para el diseño de la aplicación fueron obtenidos principalmente mediante reuniones con el cliente, en este caso el tutor del presente trabajo. De estas reuniones se extrajo que la aplicación debía ser sencilla e intuitiva para que pudiera ser utilizada por usuarios con poca o ninguna experiencia en el uso de aplicaciones similares, tenía que tener un buen rendimiento ya que de no ser así la experiencia de usuario se resentiría debido a la repetitividad del test el cual puede llegar a cansar a los usuarios. Por eso mismo era necesario proporcionar un mecanismo para que se pudiera dejar un cuestionario sin terminar y poder continua más tarde.

Era necesario que los resultados del cuestionario fueran fácilmente exportables de manera que resultara fácil al usuario compartir los resultados, por lo que se decidió un formato de informe que facilitara esta tarea.

## 3.2 Metodología

Al tratarse de un proyecto desarrollado por una única persona y que tiene un alcance limitado eminentemente académico, se ha decido emplear un desarrollo clásico o en cascada ya que esta es una de las metodologías estudiadas durante el grado de ingeniería informática.

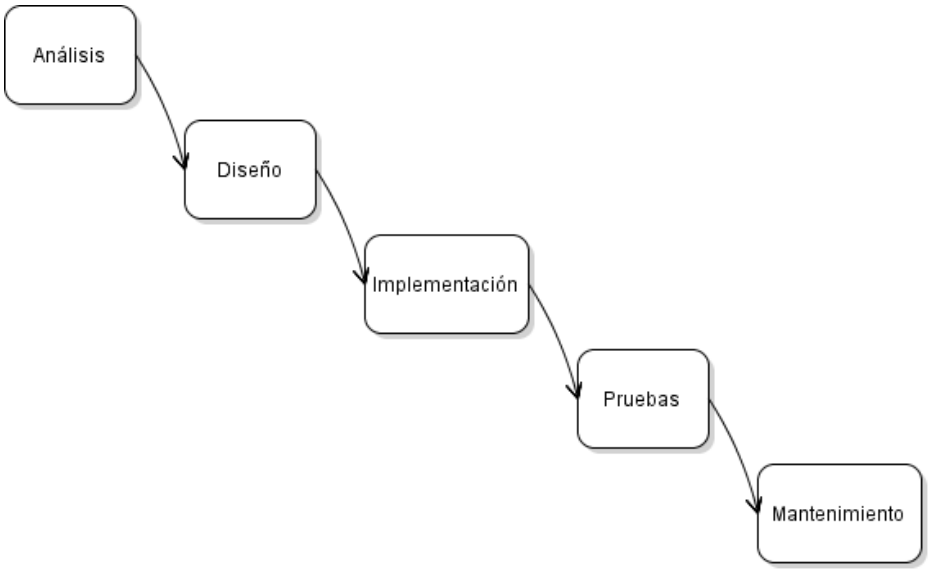


*Ilustración 3: Modelo clásico. Elaboración propia*

Como se puede observar en la ilustración dos, el modelo clásico o en cascada, establece una serie de tareas que deben realizares de forma consecutiva. “Un enfoque de cascada estricto desalienta revisar cualquier fase anterior una vez que esté completa. Esta "inflexibilidad" en un modelo de cascada puro ha sido una fuente de crítica por los partidarios de otros modelos más "flexibles".” (*Software development process*, Wikipedia, 2017 [6])

Esta rigidez del modelo clásico hace que en la práctica se transforme en uno más dinámico que permite la revisión y repetición de fases ya concluidas. De esta manera es más fácil detectar y subsanar errores.

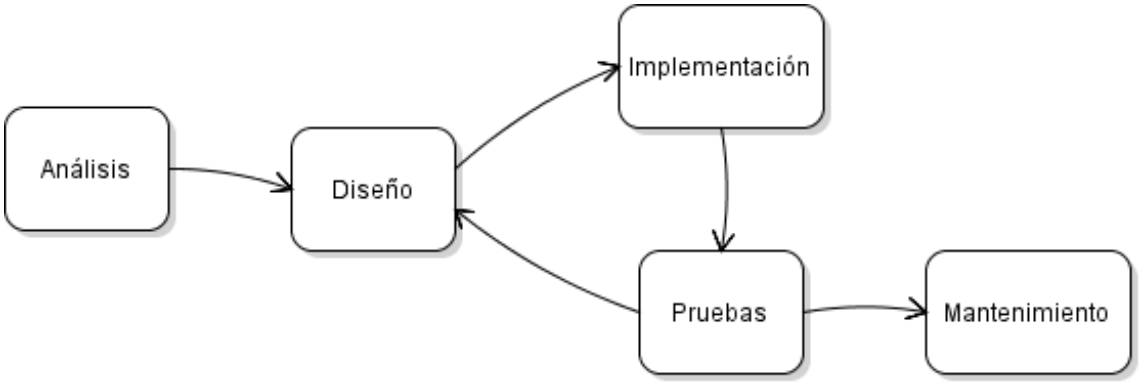


*Ilustración 4: Proceso de software, elaboración propia*

## 3.3 Tecnologías empleadas

En el desarrollo de esta aplicación se han empleado las siguientes tecnologías:

- *Java*: Lenguaje de programación orientado a objetos en el que se ha desarrollado la aplicación, está ampliamente extendido en la industria siendo usado en multitud de aplicaciones desde videojuegos, aplicaciones, webs …

- *SQLite*: Es un motor de base de datos relacionales que no necesita de un servidor dedicado para su funcionamiento, por ello es ideal para una aplicación de escritorio como esta. Además, es de código abierto y completamente gratuito.

- *JUnit*: Librería para crear pruebas unitarias para programas Java.

- *Dynamic Reports*: Librería que permite generar informes a partir de plantillas predefinidas, está basado en *Jaspe rReports*, uno de los generadores e informes más usados del mercado. Se ha usado *Dynamic Reports* y no *Jasper Reports* ya que simplifica parte del desarrollo de informes y permite generar informes de forma dinámica. Por otra parte, su falta de una adecuada documentación puede suponer un problema, sobre todo si no se tiene experiencia en este tipo de tecnología.

- *JavaFx*: Librería que permite diseñar e implementar interfaces gráficas, especialmente indicado para realizar aplicaciones que sigan el patrón modelo-vista-controlador.

- Eclipse: Entorno de desarrollo integrado escrito en Java y empleado principalmente en el desarrollo de aplicaciones usando este lenguaje de programación. Tiene una gran comunidad de usuarios y, actualmente, se distribuye bajo la Licencia Pública Eclipse la cual está considerada como una licencia de *software* libre.

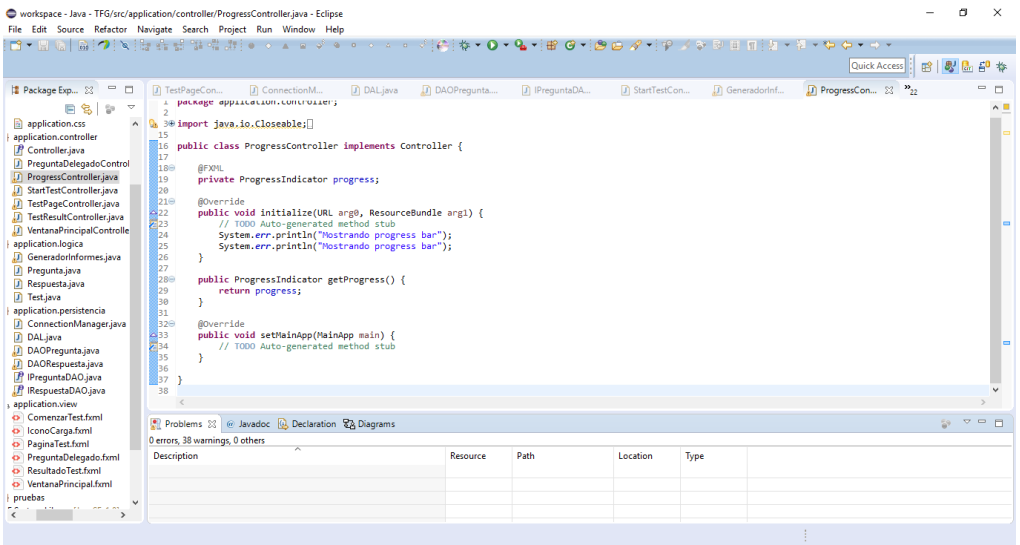


*Ilustración 5: Captura de Eclipse. Elaboración propia*

## 3.4 El cuestionario

La parte fundamental de la aplicación es el cuestionario el cual sirve para determinar si la empresa cumple o no con el nuevo reglamento europeo de protección de datos. El cuestionario está formado por treinta y cuatro preguntas que se han organizado en tres categorías: información y derechos, seguridad y delegado de protección de datos. El primer grupo, consta de preguntas relacionadas con la forma en la que la empresa se comunica con los interesados, así como de cuestiones sobre los derechos de los mismos. El grupo de peguntas sobre seguridad contiene cuestiones sobre la manera en la que la empresa maneja los posibles problemas de seguridad en el tratamiento de datos personales. Por último, se encuentran las preguntas sobre la figura del delegado de protección de datos la cual se ha introducido en el nuevo reglamento y a la que se le asigna una importancia capital en el tratamiento de datos personales.

Debido a que no todas las empresas deben disponer de un delegado de protección de datos, el bloque de preguntas relacionadas con este solo aparecerán cuando el usuario indique que su empresa tiene un delegado de protección de datos, en caso contrario estas preguntas no aparecerán en el cuestionario.

Todas las preguntas han sido redactadas de forma que se evita la ambigüedad para evitar interpretaciones incorrectas y, a la postre, fallos en el cuestionario. Resulta vital para un correcto análisis de la situación de la empresa que las respuestas al cuestionario representen realmente la realidad de la empresa por ello se ha puesto especial énfasis en que las preguntas sean claras y concisas para reducir el riesgo de que se contesten incorrectamente.

Es posible consultar un listado con todas las preguntas en el anexo (página. 56)

## 3.5 Diseño lógico

Partiendo de la base de que la funcionalidad más importante de la aplicación es el formulario, era necesario diseñar un sistema de test que fuera flexible permitiendo a los usuarios tanto retroceder para cambiar preguntas ya contestadas como guardar test que no se han completado para poder terminarlos en otro momento.

Ya que vamos a trabajar con el paradigma de la programación orientada a objetos, merece la pena ver alguno de los conceptos claves del mismo:

- Un **objeto o instancia** se puede definir como “una colección de datos y operaciones que poseen determinada estructura y mediante los cuales se modelan aspectos relevantes de un problema” (Prieto Sáez, Natividad. [et al.], 2013: 15 [5]). El estado del objeto se almacena en variables llamadas referencias [3], mientras que los métodos son los encargados de proporcionar el comportamiento. Se puede pensar en ellos como objetos del mundo real.

- Una **clase** es una plantilla para la creación de objetos, en ellas se definen las variables que almacenarán el estado del objeto y se implementan los métodos que dotarán de comportamiento a los objetos que instancien esa clase.

## 3.5.1 Arquitectura de la aplicación

### 3.5.1.1 El patrón Modelo-Vista-Controlador

Uno de los objetivos de este proyecto era conseguir una aplicación que fuera fácil de mantener y de utilizar, para conseguirlo, la aplicación ha sido diseñada usando el patrón modelo-vista-controlador (MVC). Este patrón se basa en la separación de conceptos, de manera que separa la lógica de la aplicación de la interfaz de usuario, persigue incrementar la mantenibilidad de la aplicación, facilitar la reutilización del código y, por lo tanto, conseguir un software más robusto y de mayor calidad.

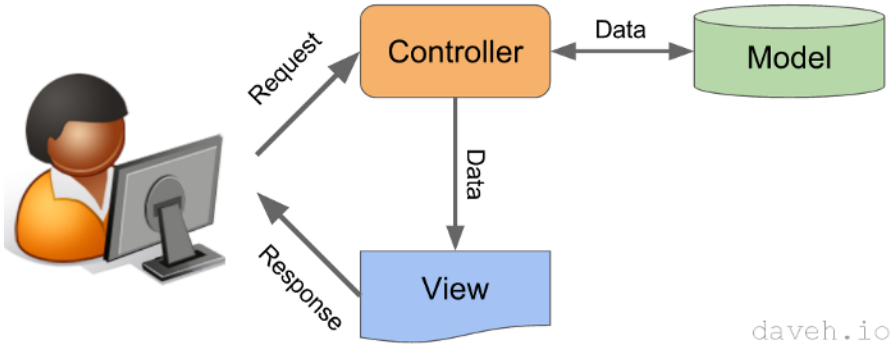


*Ilustración 6: Diagrama modelo-vista-controlador. HelloACM, https://helloacm.com/model-view-controller-explained-in-c/*

A continuación, se describe el funcionamiento del patrón modelo-vista-controlador y de los componentes que lo forman.

- Modelo: Es la parte lógica de la aplicación, la encargada acceder a los datos y realizar operaciones con ellos, los datos suelen estar almacenados en una base de datos por lo que el modelo debe ser capaz de acceder a la misma por lo que dispondrá de las funciones necesarias a tal efecto.

- Vista: Es la parte del programa que forma la interfaz gráfica, es la responsable de interactuar con los usuarios.

- Controlador: Es el responsable de responder a los eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta el 'modelo' (por ejemplo, desplazamiento o *scroll* por un documento o por los diferentes registros de una base de datos), por tanto, se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo'.

Como explica Bautista Perales, Ismael ("Aplicación Web de bases de datos usando el Framework Ruby on Rails", 2014: 13 [17]), el flujo de control que sigue un programa que implementa el MVC es el siguiente:

1. El usuario interacciona con la interfaz de usuario (pulsando un botón, por ejemplo).

2. El controlador recibe la notificación de la acción realizada por él usuario y gestiona el evento a través de un gestor de eventos (*handler [1]*).

3. El controlador realiza las operaciones necesarias sobre modelo, actualizándolo o modificándolo dependiendo de la acción realizada por el usuario.

4. La vista se actualiza obteniendo los nuevos datos del modelo.

5. La interfaz de usuario espera otra interacción del usuario con la que comenzará otro nuevo ciclo.

Con el objetivo de implementar este patrón la aplicación se ha organizado de la siguiente manera.

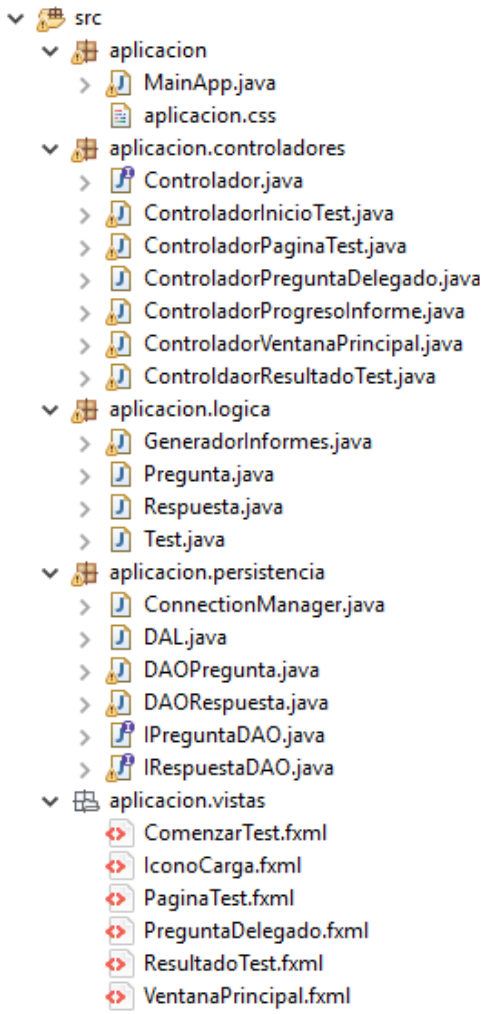


*Ilustración 7: Estructura del proyecto. Elaboración propia*

Con esta separación en paquetes, se respeta el patrón de “modelo-vista-controlador”, el modelo está formado por los paquetes “logica” y “persistencia” que serán los encargados de manipular los datos de la aplicación, el paquete “controlador” contiene todos los controladores y el paquete “vistas” contiene las vistas de la aplicación. Todos estos paquetes se encuentran dentro del paquete “aplicación” que además contiene la clase principal de la aplicación y el archivo “aplicacion.css” [8] usado para dar formato a la interfaz.

### 3.5.1.2 El patrón *Singleton*

“El patrón *Singleton* es un patrón de diseño que permite restringir la creación de objetos pertenecientes a una clase, con el objetivo de garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella” (*Singleton*. Wikipedia. 2017 [7]). Se emplea cuando ciertos datos deben estar disponible para todos los demás objetos de la aplicación.

Este patrón se ha implementado en la clase “Test”, “DAL” y “GeneradorInformes”.

A continuación, se explica la implementación de este patrón en la clase Test, aunque esta es idéntica para el resto de clases que lo implementan.

```
public static Test getReference(){
    if(reference == null){
        reference = new Test();
        return reference;
    }else{
        return reference;
    }
}
```

*Ilustración 8: Método getReference de la clase Test. Elaboración propia*

El patrón funciona de la siguiente manera, si un objeto necesita una referencia de la clase Test, realiza una llamada al método “getReference”. Este método comprueba si la variable “reference” de la clase Test es nula, de ser así crea un nuevo test y guarda una referencia a este test en la variable, a continuación, devuelve esa referencia. Si la referencia no es nula, es decir, ya se ha creado el test con anterioridad, simplemente devuelve la referencia.

De esta manera la primera vez que un objeto obtenga una referencia de la clase “Test”, se creará un nuevo objeto test, el resto de las veces se devolverá el mismo objeto por lo que durante toda la ejecución solo tendremos un único objeto test, además cualquier objeto podrá obtener una referencia al test de modo que es mucho más fácil trabajar sobre el test.

#### 3.5.1.3 El patrón Data Acces Object

Con el fin de aumentar la mantenibilidad de la aplicación, se ha aplicado el patrón *Data Acces Object* (DAO) en la capa de persistencia. Este patrón busca encapsular el acceso a la base de datos de manera que cualquier objeto que necesite acceder a la base de datos lo haga a través de un DAO. Este patrón será explicado con más profundidad en el apartado 3.7 Persistencia de datos.

## 3.6 Clases

En este apartado se describen las clases más relevantes de la aplicación. Se comenzará explicando la clase "MainApp" la cual es la clase principal de la aplicación, después se explicarán las clases de la lógica de negocio de la aplicación y, finalmente, las clases que componen los controladores de la parte gráfica de la aplicación haciendo hincapié en las clases "ControladorVentanaPrincipal" y "ControladorPaginaTest" al ser estos controladores de suma importancia para la aplicación.

### 3.6.1 Clase principal

| Clase | Atributos | Métodos |
|---|---|---|
| **MainApp** | initRootLayout<br>primaryStage | start()<br>initRootLayout()<br>abrirVista()<br>setMainApp()<br>getPrimaryStage()<br>getURL<br>main() |

La clase "MainApp" es la clase principal de la aplicación, al comenzar la ejecución se ejecuta el método "main" el cual es el punto de entrada [7] de la aplicación. Al ejecutarse este método, se crea un nuevo objeto de tipo test, y se recuperan las preguntas almacenadas en la base de datos. Las preguntas se almacenan en las colecciones que posteriormente se le pasan al test. De esta forma el objeto test ya dispone de las preguntas que formarán el cuestionario.

```java
public static void main(String[] args) {

    DAL dal = null;

    try {
        dal = DAL.getReference();
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    Test test = Test.getReference();

    //Recuperar las perguntas de la base de datos
    ArrayList<Pregunta> preguntas = new ArrayList<Pregunta>();
    preguntas = dal.obtenerPreguntas();

    ArrayList<Pregunta> preguntasDPD = new ArrayList<Pregunta>();
    preguntasDPD = dal.obtenerPreguntasDPD();

    //Asignamos las preguntas al test
    test.setListaPreguntas(preguntas);
    test.setPreguntasTestDPD(preguntasDPD);
    test.setIndicePreguntas(0);
    launch(args);
}
```

*Ilustración 9: Método main de la clase MainApp. Elaboración propia*

Una vez se ha terminado de configurar el objeto test, se realiza una llamada al método "launch" el cual inicializa la interfaz gráfica usando los métodos "start" e "initRootLayout". Este último configura la venta para tener una dimensión de 800x600 píxeles y carga la vista definida en el archivo "VentanaPrincipal.fxml". Para cargarla, se crea una instancia de la clase "FXMLoader" que permite cargar una jerarquía de objetos a partir de un archivo *FXML* [5], tras establecer la ruta en la que se encuentra el archivo "VentanaPrincipal.fxml", podemos cargar la información de la vista.

El siguiente paso es obtener el controlador asociado a la vista y usando el método "setMainApp" del mismo se le asigna una referencia a la clase principal. De este modo desde el controlador podremos acceder a la clase "MainApp" para cambiar entre ventanas. Por último, se crea una nueva escena usando la información de la vista que hemos cargado anteriormente, se le asigna la hoja de estilo "aplicacion.css", se establece la escena y se muestra.

```
public void initRootLayout() {
    try {
        primaryStage.setHeight(600);
        primaryStage.setWidth(800);
        // Load root layout from fxml file.
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(getURL("view/VentanaPrincipal.fxml"));
        rootLayout = (BorderPane) loader.load();
        rootLayout.getStyleClass().add("borderPane");
        Controller controller = loader.getController();
        controller.setMainApp(this);

        // Show the scene containing the root layout.
        Scene scene = new Scene(rootLayout);
        scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());

        primaryStage.setScene(scene);
        primaryStage.show();

    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

*Ilustración 10: Método InitRootLayout de la clase MainApp. Elaboración propia*

La clase "MainApp" dispone de un método que permite cambiar la vista que se muestra al usuario. Este método llamado "abrirVista", recibe como parámetro [4] el nombre del archivo FXML que queremos abrir. Por ejemplo, si deseamos mostrar la ventana principal, cuando llamemos al método "abrirVista" le pasaremos como parámetro "VentanaPrincipal.fxml".

```
public void abrirVista(String urlVista) {
    FadeTransition fadeTransition = new FadeTransition(Duration.millis(250),
            primaryStage.getScene().getRoot());
    fadeTransition.setFromValue(1.0);
    fadeTransition.setToValue(0.0);
    fadeTransition.setOnFinished(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event) {
             try {
               // Load view
               FXMLLoader loader = new FXMLLoader();
               loader.setLocation(MainApp.class.getResource(urlVista));
               BorderPane vista = (BorderPane) loader.load();

               Controller controller = loader.getController();
               setMainApp(controller);
               // Set view into the center of root layout.
               Scene scene = new Scene(vista);
               primaryStage.setScene(scene);
               scene.getStylesheets().add(getClass().
                       getResource("application.css").toExternalForm());
               vista.getStyleClass().add("borderPane");
            }catch (IOException e) {
                e.printStackTrace();
            }
        }
    });
    fadeTransition.play();
}
```

*Ilustración 11: Método abrirVista. Elaboración propia*

Como se observa en la ilustración once, el método "abrirVista" crea una transición para que el paso de una ventana a otra se produzca de una manera más fluida y agradable. La transición es un fundido a negro con una duración de dos ciento cincuenta milisegundos una vez terminada la cual se carga la nueva vista. El proceso para cargar una nueva vista es el mismo que se ha explicado anteriormente.

## 3.6.2 Clases de la lógica

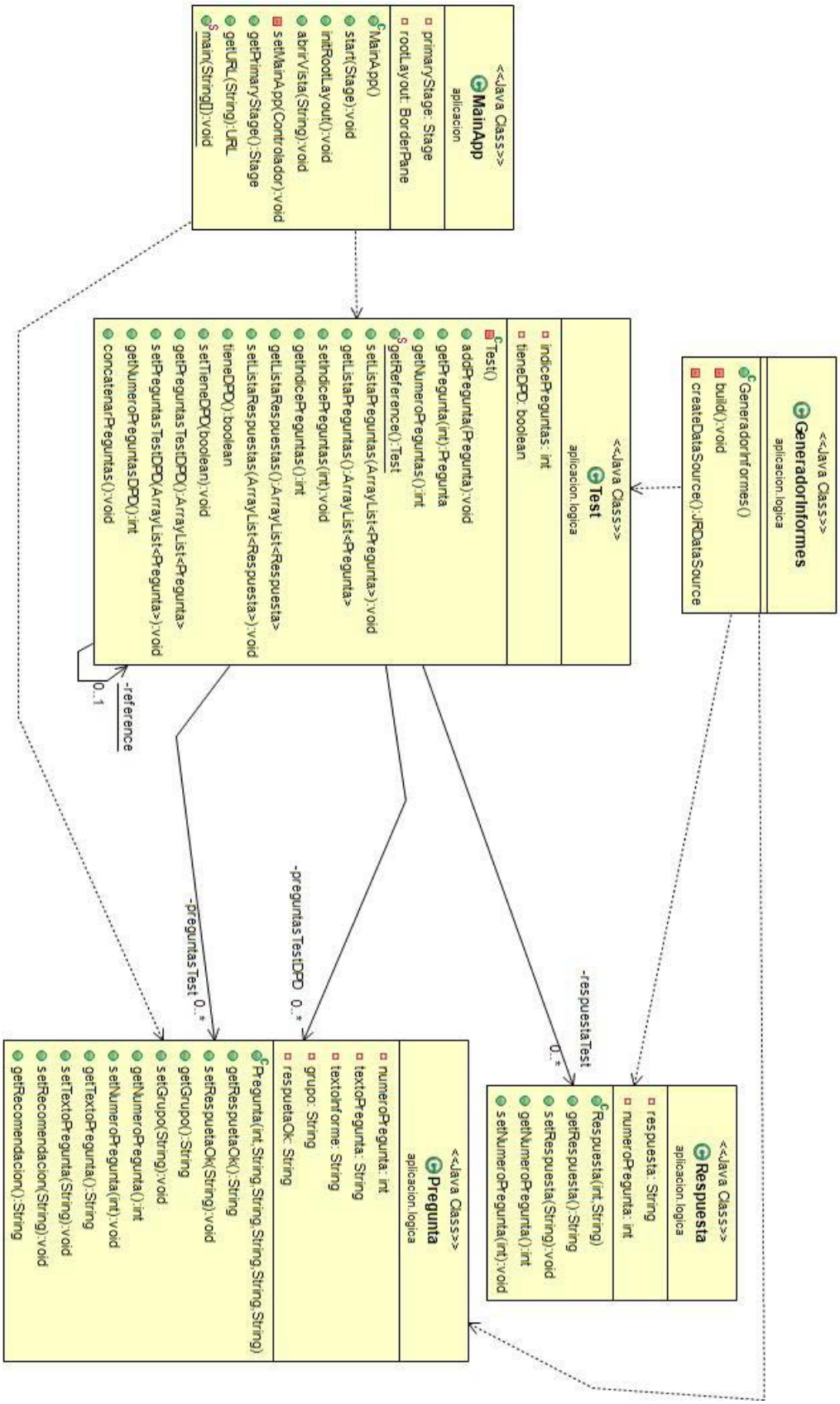


*Ilustración 12: Diagrama de clases de la capa de lógica. Elaboración propia*

| Clase | Atributos | Métodos |
|---|---|---|
| **Test** | preguntasTest<br>preguntasTestDPD<br>respuestasTest<br>reference<br>indicePreguntas<br>tieneDPD | setPreguntasTest()<br>getPreguntasTest()<br>setPreguntasTestDPD()<br>getPreguntasTestDPD()<br>setRespuestasTest()<br>getRespuestasTest()<br>getReference()<br>getIndicePreguntas()<br>setIndicePreguntas()<br>tieneDPD()<br>setTieneDPD()<br>concatenarPreguntas() |
| **Pregunta** | numeroPregunta<br>textoPregunta<br>texotInforme<br>grupo<br>repuestaOk | setNumeroPregunta()<br>gerNumeroPregunta()<br>setTextoPregunta()<br>getTextoPregunta()<br>setTextoInforme()<br>getTextoInforme()<br>setGrupo()<br>getGrupo()<br>setRespuestaOk()<br>getRespuestaOk() |
| **Respuesta** | respuesta<br>numeroPregunta | setRespuesta()<br>getRespuesta()<br>setNumeroPregunta()<br>getNumeroPregunta() |
| **GeneradorInformes** | | build()<br>createDataSource() |

### 3.6.2.1 Clases Pregunta y Respuesta

La clase “Pregunta” define las preguntas que forman el cuestionario, cada pregunta tiene un número que la identifica, un texto de la pregunta, el texto que aparecerá en el informe en caso de que no se conteste de forma correcta a la misma, el nombre del grupo al que pertenece la pregunta y cuál es la respuesta que se considera correcta a la pregunta.

La clase “Respuesta” es el modelo para las respuestas al cuestionario, solo tiene un número que se corresponde con el número de la pregunta a la cual pertenece la respuesta y la respuesta que ha dado el usuario (si o no).

### 3.6.2.2 Clase Test

La clase “Test” es el modelo que utiliza la aplicación para implementar la consulta. Como se ha visto en el apartado dedicado al cuestionario, este es una parte fundamental de la aplicación. Para modelarlo, la clase “Test” consta de una colección para todas las preguntas que no tienen relación con la figura del delegado de protección de datos (preguntasTest), una colección para las preguntas que están relacionadas con el delegado de protección de datos (preguntasTestDPD), una colección de Respuestas (respuetasTest), un entero que indica cual es la pregunta actual (indicePreguntas) y una referencia a si misma (test).

La existencia de dos colecciones para las preguntas se debe al hecho de que las preguntas sobre el delegado de protección de datos puede que no deban aparecer en el cuestionario. De esta manera, si no se quiere mostrar al usuario dichas preguntas, se utiliza únicamente la colección “preguntasTest”, mientras que, si se desea incluir las preguntas sobre el delegado de protección de datos, se utiliza el método “concatenarPreguntas” para añadir a “preguntasTest” las preguntas de la colección “preguntasTestDPD”. De este modo obtenemos una lista con todas las preguntas que forman el cuestionario.

### 3.6.2.3 Clase GeneradorInformes

```
private void build(){
    TextColumnBuilder<String> itemColumn = col.column("", "item", type.stringType());

    ColumnGroupBuilder itemGroup = grp.group(itemColumn)
     .setTitleWidth(30)
     .setHeaderLayout(GroupHeaderLayout.TITLE_AND_VALUE)
     .showColumnHeaderAndFooter();

    StyleBuilder boldStyle         = stl.style().bold().setFontSize(24);
    StyleBuilder boldCenteredStyle = stl.style(boldStyle).setHorizontalAlignment(HorizontalAlignment.CENTER);


    try{
        report()
        .setTemplate(Templates.reportTemplate)
        .setShowColumnTitle(false)
        .columns(
                col.column("", "Recomendaciones", type.stringType()))
        .groupBy(itemGroup)
        //.title(Templates.createTitleComponent("Informe de Resultados"))
        .title(cmp.text("Informe de resultados").setStyle(boldCenteredStyle))
        .pageFooter(Templates.footerComponent)
        .setDataSource(createDataSource())
        .show(false);
    }catch(Exception e){
        e.printStackTrace();
    }

}
```

*Ilustración 13: Método build de la clase GeneradorInformes. Elaboración Propia*

La clase “GeneradorInformes” es la encargada de generar el informe a partir de los resultados del test. En esta clase también se implementa el patrón *Singleton*. Al crear una instancia de esta clase, se ejecuta el método “build” el cual configura el informe usando los métodos proporcionados por la librería *Dynamic Reports*.

Se crea el informe y se establece la plantilla que servirá de base para el mismo, se quita el título de las columnas y se añaden estas al informe, en este caso, el informe consta de una única columna con las recomendaciones. Se establece el título "Informe de resultados", y se asigna la fuente de datos del informe, es decir, de donde se van a obtener los datos que se usarán para formar el documento. En nuestro caso, creamos una fuente de datos que obtiene los datos del test que ha realizado el usuario de la aplicación.

```
private JRDataSource createDataSource() {
    DRDataSource dataSource = new DRDataSource("item","Recomendaciones");
    Test test = Test.getReference();
    ArrayList<Respuesta> respuestas = test.getListaRespuestas();
    ArrayList<Pregunta> preguntas = test.getListaPreguntas();
    for(int i=0; i < respuestas.size(); i++){
        if(!respuestas.get(i).equals(preguntas.get(i).getRespuetaOk())){
            dataSource.add(preguntas.get(i).getGrupo(),preguntas.get(i).getRecomendacion());
        }
    }
    return dataSource;
}
```

*Ilustración 14: Método createDataSource de la clase GeneradorInformes. Elaboración propia*

El método "createDataSource" crea una fuente de datos (*data source [6]*) usando las respuestas del test, para ello, tras obtener una referencia al test, se recorren las listas de respuestas y de preguntas y se comprueba cada respuesta dada por el usuario con la que se considera correcta para esa pregunta, si la respuesta es incorrecta se añade la recomendación asociada a esa pregunta a la fuente de datos. Además, se usa el grupo al cual pertenece la pregunta para agrupar las recomendaciones de modo que el informe queda dividido en tres secciones una para cada tipo de preguntas.

El resultado es un informe en el que, divididas en categorías, aparecen las recomendaciones de las preguntas que el usuario ha contestado erróneamente. Este informe puede ser exportado a multitud de formatos como *PDF*, *DOCX*, *HTML* entre otros. De esta forma el usuario puede hacer llegar de una manera rápida y cómoda el informe al responsable de aplicar las medidas oportunas para subsanar las deficiencias halladas.

## 3.6.3 Controladores

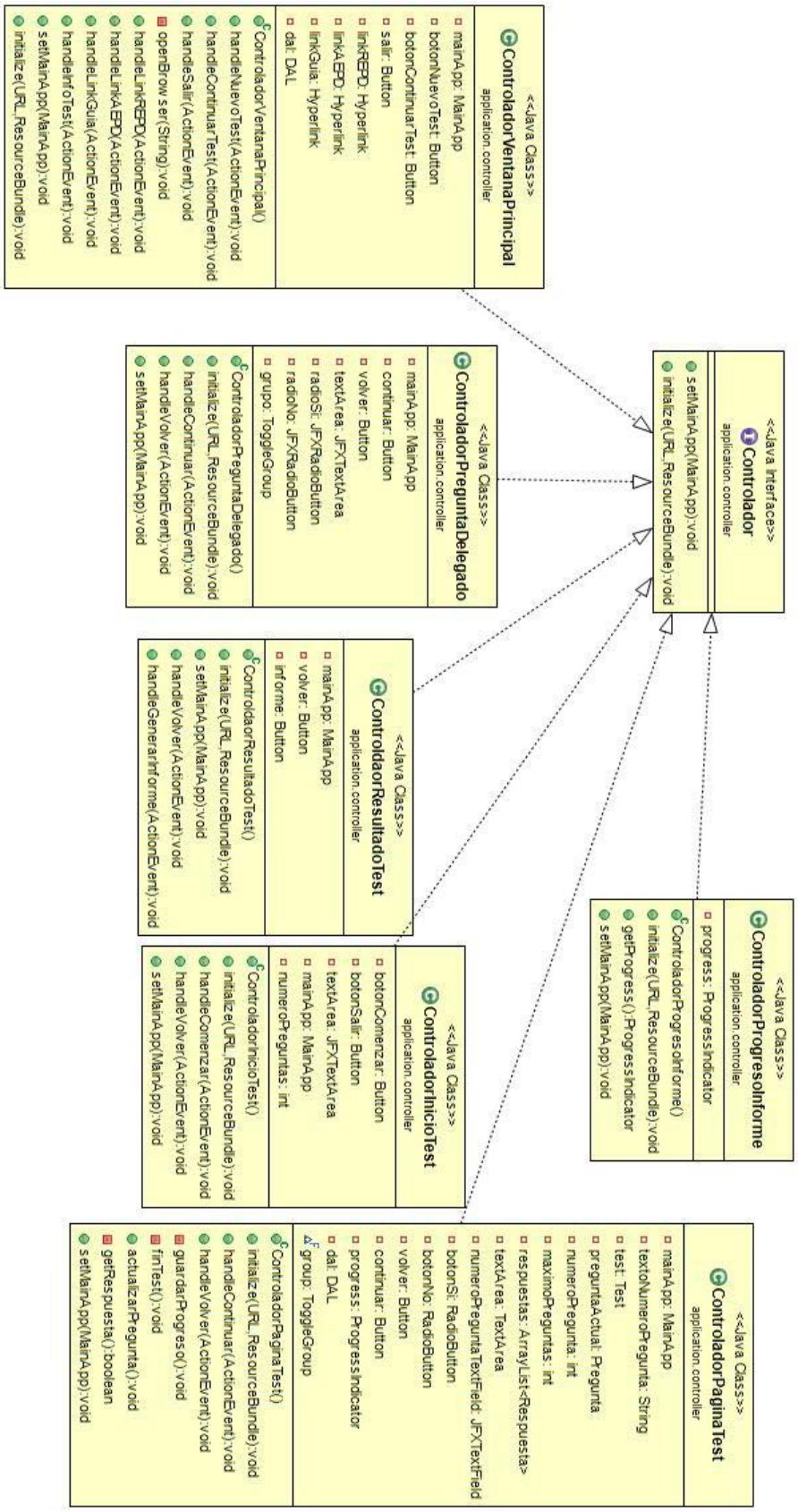


*Ilustración 15: Diagrama de clases de los Controladores. Elaboración propia*

| Clase | Atributos | Métodos |
|---|---|---|
| **ControladorPreguntaDelegado** | continuar<br>volver<br>textArea<br>radioSi<br>radioNo<br>grupo | initialize()<br>setMainApp<br>handleContinuar()<br>handleVolver() |
| **ControladorProgresoInforme** | mainApp<br>progressBar | initialize()<br>getProgressBar |
| **ControladorInicioTest** | mainApp<br>botonComenzar<br>botonVolver<br>textArea | initialize()<br>setMainApp()<br>handleComenzar()<br>handleVolver() |
| **ControladorPaginaTest** | mainApp<br>textArea<br>numeroPreguntaTextField<br>botonSi<br>botonNo<br>volver<br>continuar<br>progress<br>textoNumeroPregunta<br>test<br>preguntaActual<br>numeroPregunta<br>maximoPreguntas<br>respuestas<br>dal | initialize()<br>setMainApp()<br>handleContinuar()<br>handleVolver()<br>guardarProgreso()<br>finTest()<br>actualizarPregunta()<br>getRespuesta() |
| **ControladorResultadoTest** | mainApp<br>volver<br>informe | initialize()<br>setMainApp()<br>handleGenerarInforme()<br>handleVolver() |
| **ControladorVentanaPrincipal** | mainApp<br>botonNuevoTest<br>botoContinuarTest<br>salir<br>linkREPD<br>linkAEPD<br>linkGuia<br>dal | initialize()<br>setMainApp()<br>handleContinuarTest<br>handleTest<br>handleSalir<br>handleLinkREPD<br>handleLinkAEPD<br>handleLinkGuia<br>openBrowser() |

Cómo ya se ha visto, en el patrón modelo-vista-controlador, los controladores son los responsables de comunicar la vista con el modelo respondiendo a las acciones que lleve a cabo el usuario y realizando las operaciones que sean necesarias.

En esta sección se va a explicar el funcionamiento de los controladores describiéndose en profundidad los controladores “ControladorVentanaPrincipal” y “ControladorPaginaTest” los cuales son los más relevantes de la aplicación.

```java
public interface Controlador extends Initializable {

    public void setMainApp(MainApp main);

    @Override
    public void initialize(URL location, ResourceBundle resources);

}
```

*Ilustración 16: Interfaz Controlador. Elaboración propia*

Todos los controladores de la aplicación implementan la interfaz "Controlador". Esta interfaz consta de dos métodos: el método "initialize" que proporciona la librería JavaFX y el método "setMainApp". La función de esta interfaz es que homogeneizar todos los controladores al obligarles a implementar los métodos mencionados anteriormente. De esta manera nos aseguramos de que todos los controladores tendrán los métodos "initialize" y "setMainApp" lo cual proporciona gran flexibilidad a la hora de trabajar con los controladores ya que cada controlador tiene su propia implementación de estos métodos al mismo tiempo que nos permite trabajar como si todos los controladores fueran iguales.

El método "initialize" es llamado al cargar la ventana de forma automática, en él se realiza la configuración del controlador estableciendo que método del controlador debe lanzarse con cada evento, así como cualquier otra operación que sea necesaria realizar cuando se abre la ventana.

```java
public class ControldaorResultadoTest implements Controlador {

    private MainApp mainApp;


    @FXML
    private Button volver;
    @FXML
    private Button informe;

    @Override
    public void initialize(URL arg0, ResourceBundle arg1) {

        volver.setOnAction(this::handleVolver);
        informe.setOnAction(this::handleGenerarInforme);
        informe.setDisable(false);
    }
```

*Ilustración 17: Clase ControladorResultadoTest. Elaboración propia*

La ilustración diecisiete muestra un fragmento del controlador "ControladorResultadoTest", en el podemos ver el método "initialize" en el cual se establecen los manejadores de eventos para cada botón de la pantalla. En concreto, cuando el usuario pulse el botón "volver" se lanzará el método "handleVolver" mientras que para el botón "informe" se lanzará "handleGenerarInforme". Por último, se deshabilita el botón informe.

Como se puede observar, la utilización del método “initialize” resulta fundamental para inicializar el controlador y la vista de forma correcta. De esta manera, gracias a JavaFX podemos inicializar de forma rápida, sencilla y automática los controladores de nuestra aplicación.

### 3.6.3.1 ControladorPreguntaDelegado

Este controlador se encarga de manejar la ventana “PreguntaDelegado” en la cual se pregunta al usuario si su empresa tiene o no un delegado de protección de datos. Dependiendo de la respuesta dada por el usuario se configura el cuestionario para realizar, o no, preguntas sobre el delegado de protección de datos.

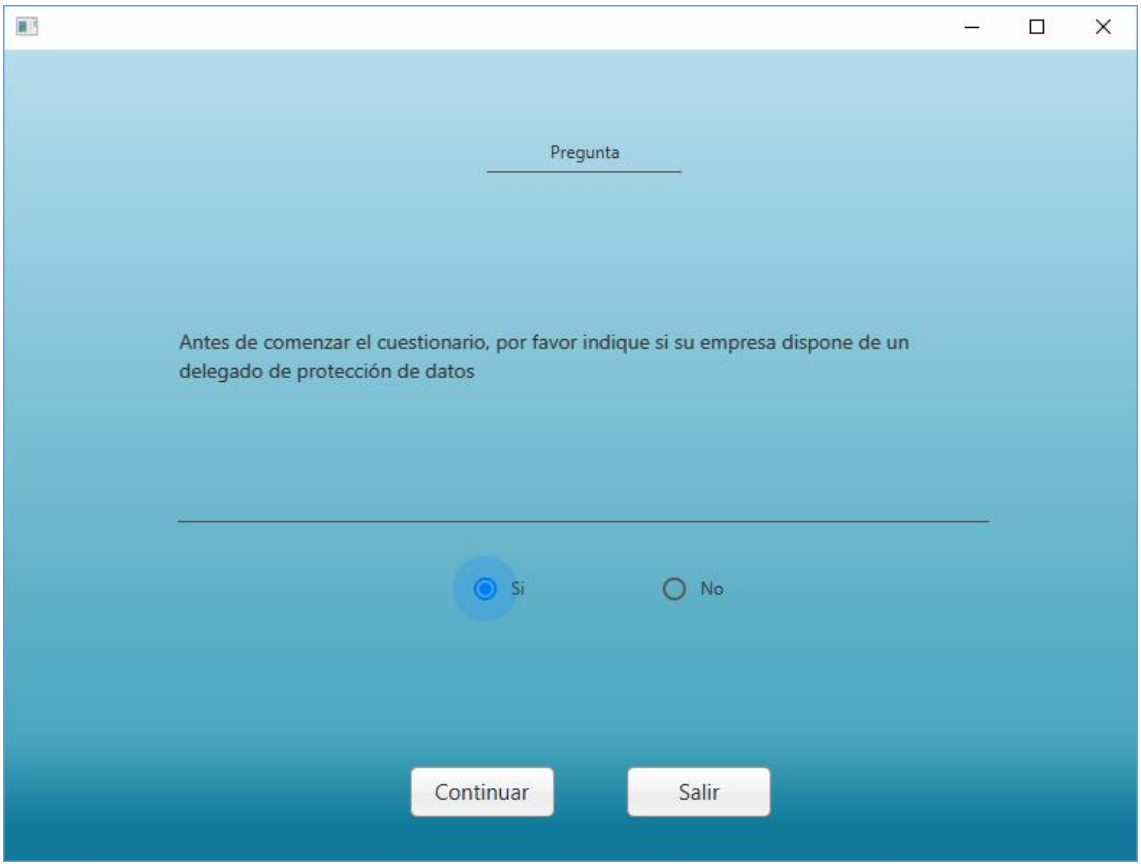


*Ilustración 18: Ventana "PreguntaDelegado". Elaboración propia*

Al abrirse la ventana, el método “intialize” establece el texto que se mostrará en la misma, en este mismo método se crea un grupo para poder manejar los botones de respuesta y se asigna a los botones “Continuar” y “Salir” sus respectivos manejadores de eventos.

```java
@Override
public void initialize(URL location, ResourceBundle resources) {
    // TODO Auto-generated method stub
    String texto ="Antes de comenzar el cuestionario,"
            + " por favor indique si su empresa dispone de"
            + " un delegado de protección de datos";
    textArea.setText(texto);

    grupo = new ToggleGroup();

    radioSi.setToggleGroup(grupo);
    radioNo.setToggleGroup(grupo);

    continuar.setOnAction(this::handleContinuar);
    volver.setOnAction(this::handleVolver);
}
```

*Ilustración 19: Método initialize de la clase ControladorPreguntaDelegado. Elaboración propia*

El método "handleContinuar" se encatga de manejar los eventos relacionados con el botón "Continuar". De manera que, cada vez que el usuario pulsa dicho botón, el método comprueba si ha seleccionado una respuesta, de no ser así, se crea una alerta en la que se informa al usuario que debe seleccionar una respuesta antes de continuar.

Si el usuario si ha seleccionado una respuesta, en caso de ser afirmativa, se modifica el atributo "tieneDPD" de la clase test a *true* en caso contrario, se modifica dicho atributo a false. De esta manera el test queda configurado y ya se puede dar comienzo con el mismo.

```java
public void handleContinuar(ActionEvent event){
    if(grupo.getSelectedToggle() != null){
        if(grupo.getSelectedToggle().equals(radioSi)){
            System.out.println("tiene dpd");
            Test.getReference().setTieneDPD(true);
        }else{
            System.out.println("no tiene dpd");
            Test.getReference().setTieneDPD(false);
        }
        mainApp.abrirVista("vistas/PaginaTest.fxml");
    }else{
        Alert alert = new Alert(AlertType.INFORMATION);
        alert.setTitle("Atención");
        alert.setHeaderText("Seleccione una respuesta para continuar");
        alert.showAndWait();
    }
}
```

*Ilustración 20: Manejador de eventos handleContinuar. Elaboración propia.*

Por último, el manejaor "handleVolver" abre la vista del menú principal utilizando el método "abriVista" de la clase MainApp cuando el usuario pulsa sobre el botón "volver". Al realiza esta acción, es indiferente si ha marcado alguna respuesta ya que estas no se tienen en cuenta. Al pulsar el botón se informa al usuario de este hecho y se le pide que confirme la acción.

### 3.6.3.2 ControladorInicioTest

Este controlador se encarga de manejar la ventana “StartTest” en la cual se da al usuario información sobre cómo debe realizar el cuestionario.

```
@Override
public void initialize(URL location, ResourceBundle resources) {
    // TODO Auto-generated method stub
    numeroPreguntas = Test.getReference().getNumeroPreguntas() + Test.getReference().getNumeroPreguntasDPD();
    String texto ="Va a comenzar el test  para comprobar que cambios debe relaizar su empresa para "
            + "adaptarse al nuevo reglamento europeo de protección de datos."
            + " El test consta de "+numeroPreguntas+" preguntas. Para cada pregunta "
                    + "seleccione la respuesta que considera adecuada,"
            + " al finalizar el test, se generará un informe recopilando las "
            + "deficencias detectadas, puede usar este informe para llevar a cabo"
            + "las reformas necesarias con el fin de subsanar dichas deficiencias."
            + " En cualquier momento puede usar el botón de volver para guardar"
            + " el progreso realizado y salir del test.";
    textArea.setText(texto);
    textArea.setEditable(false);

    botonVolver.setOnAction(this::handleVolver);
    botonComenzar.setOnAction(this::handleComenzar);
}
```

*Ilustración 20: Método initialize. Elaboración propia.*

En el método “initialize” se establece el texto que debe mostrarse al abrirse la ventana, así como los manejadores de los botones “Comenzar” y “Salir”. Los manejadores “handleVolver” y “handleComenzar” se encargan de gestionar los eventos relacionados con dichos botones. Si el usuario pulsa el botón volver, la aplicación carga la vista del menú principal mientras que, si pulsa continuar, se carga la vista “PreguntaDelegado” y, de esta manera da comienzo el cuestionario.

### 3.6.3.3 ControladorResultadoTest

Este controlador se encarga de manejar la ventana “TestResult” la cual se abre al finalizar el cuestionario. En ella el usuario puede volver a generar el informe o volver al menú principal.

El método “initialize” ha sido visto al inicio de este apartado por lo que solo queda por mencionar los manejadores de eventos de los que dispone el controlador. Se trata de los métodos “handleVolver” y “handleGenerarInforme”. El primero de ellos se ejecuta cuando el usuario pulsa sobre el botón “volver”, se pregunta al usuario si está seguro de que desea salir al menú principal ya que esta acción conllevará la pérdida del informe generado. Si el usuario responde afirmativamente, se abre la vista “VentanaPrincipal” usando el método “abrirVista” de la clase “MainApp”. El segundo manejador se ejecuta cuando el usuario pulsa sobre el botón “generar informe” creando un nuevo informe con los datos del cuestionario.

### 3.6.3.4 ControladorVentanaPrincipal

Es el controlador de la ventana principal de la aplicación, dispone de los métodos necesarios para comenzar un nuevo test, continuar uno en caso de que existan respuestas guardadas en la base de datos, así como para salir de la aplicación y abrir en el navegador los links de interés.


*Ilustración 21: Ventana principal de la aplicación. Elaboración propia*

Como hemos visto anteriormente el método "initialize" se encarga de la inicialización del controlador, en este caso se obtiene una instancia de la clase DAL la cual nos proporciona métodos para acceder a la base de datos y poder realizar operaciones sobre ella, esta clase se explicará con más detalle en el apartado de persistencia.

Después de obtener la instancia de la clase DAL, se usa para averiguar si hay respuestas guardadas en la base de datos, de ser así, se habilita el botón de continuar test para que el usuario pueda continuarlo. Si no hay respuesta, el botón permanece deshabilitado.

A continuación, se asocian los métodos controladores con sus correspondientes botones y con los enlaces.

```java
@Override
public void initialize(URL location, ResourceBundle resources) {
    try{
        this.dal = DAL.getSingleton();
    }catch(Exception e){
        e.printStackTrace();
    }
    if(!dal.hayRespuestas()){
        botonContinuarTest.setDisable(true);
    }else{
        botonContinuarTest.setDisable(false);
    }
    botonNuevoTest.setOnAction(this::handleNuevoTest);
    botonContinuarTest.setOnAction(this::handleContinuarTest);
    salir.setOnAction(this::handleSalir);
    linkREPD.setOnAction(this::handleLinkREPD);
    linkAEPD.setOnAction(this::handleLinkAEPD);
    linkGuia.setOnAction(this::handleLinkGuia);
}
```

*Ilustración 22: Método initialize de la clase ControladorVentanaPrincipal. Elaboración propia*

```java
@FXML
public void handleNuevoTest(ActionEvent event){
    //Iniciar un nuevo Test
    System.out.println("Nevo test");
    try {
        dal.borrarRespuestas();
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    mainApp.abrirVista("vistas/ComenzarTest.fxml");
}
```

*Ilustración 23: Método handleNuevoTest de la calse ControladorVentanaPrincipal. Elaboración propia*

"handleNuevoTest": Inicia un nuevo test abriendo la ventana "startTest" y elimina las respuestas de la base de datos en caso de haberlas. Ya que todos los test son iguales no tiene sentido guardar repuestas que pertenezcan a varios cuestionarios. Para borrar las respuestas se utiliza la instancia de la clase DAL que se obtiene en el método "initialize". Tras borrar las respuestas se abre la ventana "ComenzarTest" usando el método "abrirVista" de la clase "MainApp".

```
@FXML
public void handleContinuarTest(ActionEvent event){
    //Iniciar un nuevo Test
    System.out.println("Continuar test");
    //Cargar preguntas ya completadas;
    try {
        Test test =  Test.getReference();

        ArrayList<Respuesta> respuestas = this.dal.obtenerRespuestas();

        test.setListaRespuestas(respuestas);
        test.setIndicePreguntas(respuestas.size());

    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    mainApp.abrirVista("vistas/ComenzarTest.fxml");
}
```

*Ilustración 24: Método handleContinuarTest de la clase ControladorVentanaPrincipal. Elaboración propia*

"handleContinuarTest": Continua un test anteriormente guardado en la base de datos, para ello se obtiene una referencia al objeto test y se recuperan las preguntas de la base de datos. Una vez tenemos la colección con las preguntas se usa el método "setListaPreguntas" para asignárselas al test. Al índice, que marca por qué pregunta se ha de comenzar el test, se le da el valor del número de respuestas que tengamos de modo que si hemos recuperado cinco preguntas el índice valdrá cinco por lo que el test comenzará por la pregunta que ocupe la quinta posición en la lista de preguntas.

```
@FXML
public void handleSalir(ActionEvent event){
    Alert alert = new Alert(AlertType.CONFIRMATION);
    alert.setTitle("Diálogo de confirmación");
    alert.setHeaderText("Va a cerrarla aplicación");

    Optional<ButtonType> result = alert.showAndWait();
    if (result.get() == ButtonType.OK){
        //Cerrar la aplicaión
       System.exit(0);
    } else {
        //cerrar el dialogo
        alert.close();
    }
}
```

*Ilustración 25: Método handleSalir de la clase ControladorVentanaPrincipal. Elaboración propia*

"handleSalir": Este método se ejecuta cuando el usuario pulsa el botón de salir. Para evitar que el usuario cierre la aplicación por error, al pulsar se crea una ventana de aviso en la que se le pide que confirme si realmente quiere cerrar la aplicación. Si responde afirmativamente la aplicación se cierra, en caso contrario, la aplicación no se cierra y el usuario puede continuar usándola.

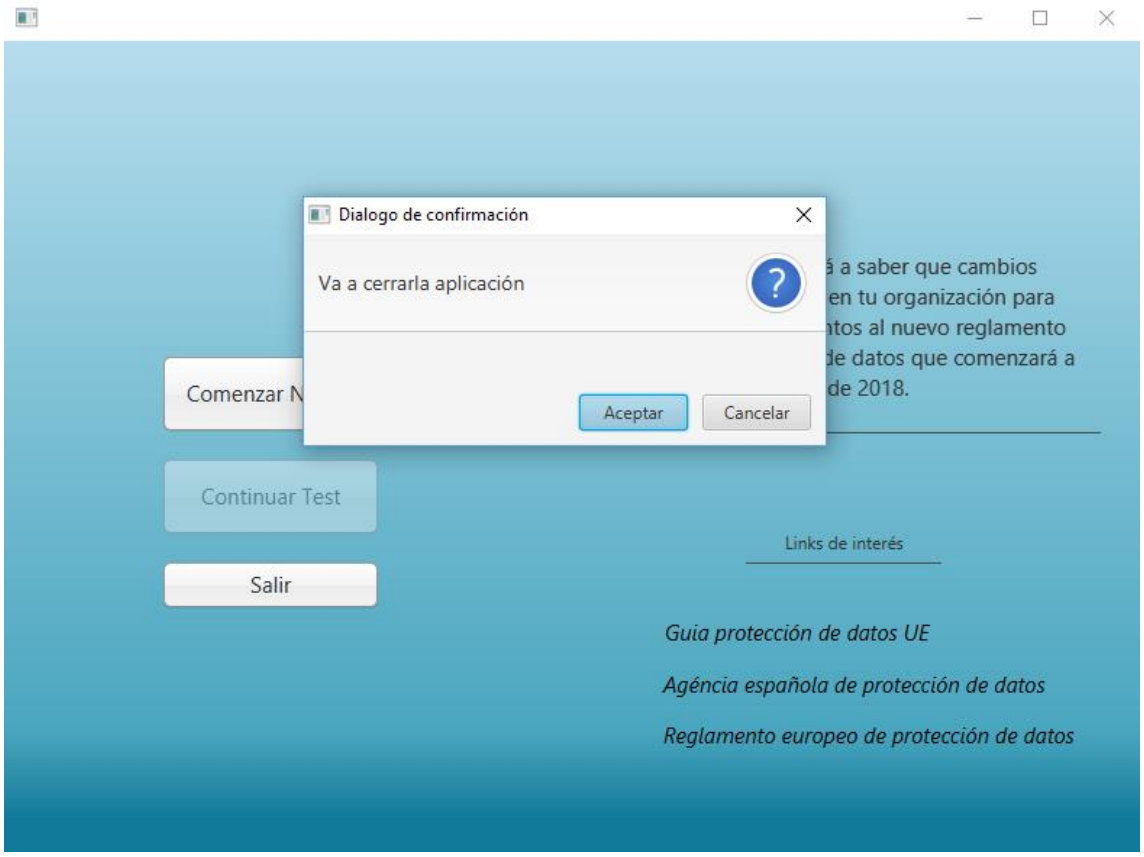


*Ilustración 26: Aviso al intentar cerrar la aplicación. Elaboración propia*

Los métodos “handleLinkREPD”, “handleLinkAEPD” y “handleLinkGuia” abren enlaces al reglamento europeo de protección de datos [12], a la agencia española de protección de datos [8] y a la guía para el nuevo reglamento [9] respectivamente. De esta manera, el usuario puede acceder a información de gran utilidad de una manera rápida.

### 3.6.4.5 ControladorPaginaTest

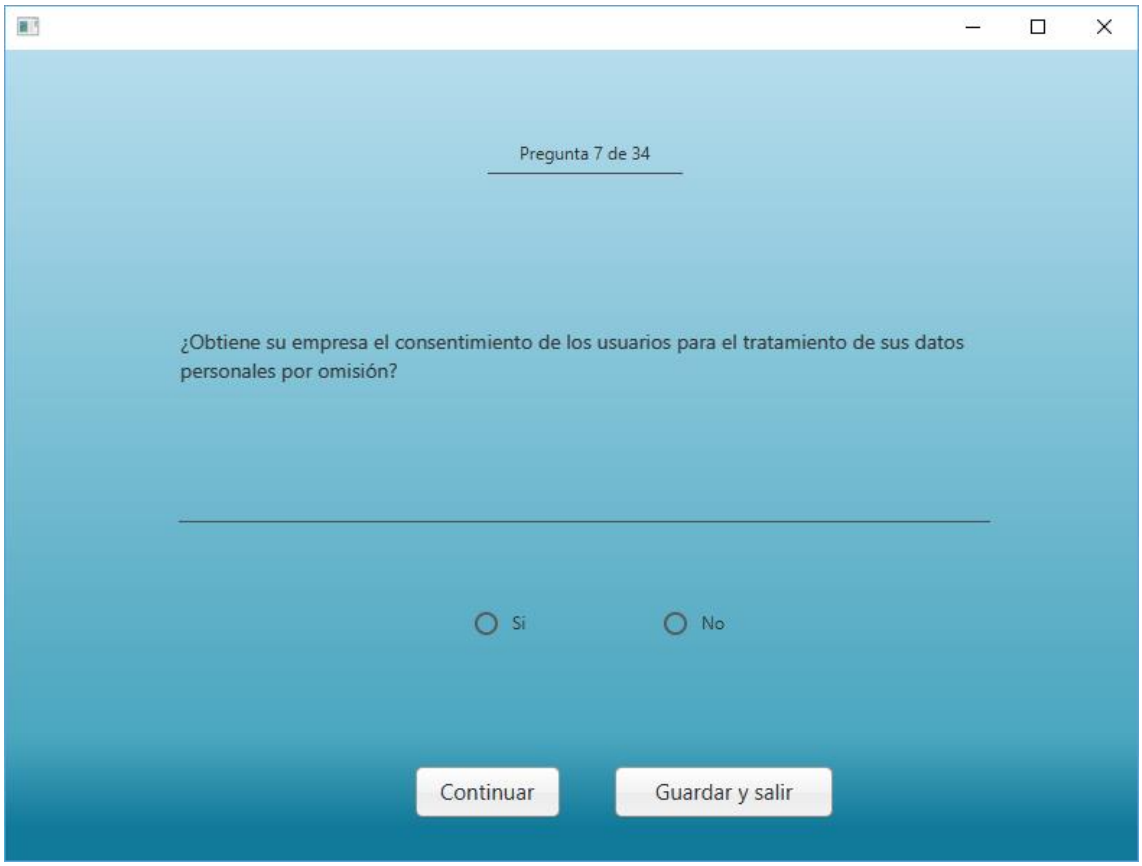


*Ilustración 27: Ventana del cuestionario. Elaboración propia*

El controlador "ControladorPaginaTest" se encarga de manejar el funcionamiento de la pantalla "PaginaTest" en la cual se muestran las preguntas que forman el cuestionario. Esta pantalla de dos botones, el botón continuar el cual se usa para pasar a la siguiente pregunta una vez se ha contestado la actual y el botón volver el cual sirve para salir del test y regresar al menú principal.

Este controlador es de suma importancia ya que controla el desarrollo del test, es el encargado de mostrar la pregunta, actualizar el número de la pregunta actual, registrar la respuesta dada por el usuario y de manejar los eventos de los dos botones.

Veamos primero como se inicializa el controlador.

```
@Override
public void initialize(URL location, ResourceBundle resources) {

    try {
        dal = DAL.getReference();
    } catch (Exception e) {
        e.printStackTrace();
    }

    continuar.setOnAction(this::handleContinuar);
    volver.setOnAction(this::handleVolver);

    this.test = Test.getReference();
    if(test.tieneDPD()){
        //incluir preguntas sobre le delegado de protección de datos.
        test.concatenarPreguntas();
    }

    this.respuestas = test.getListaRespuestas();
    maximoPreguntas = test.getNumeroPreguntas();
    preguntaActual = test.getPregunta(test.getIndicePreguntas());
    numeroPregunta = preguntaActual.getNumeroPregunta();
    textoNumeroPregunta = "Pregunta " +numeroPregunta+" de "+ maximoPreguntas;
    numeroPreguntaTextField.setText(textoNumeroPregunta);
    textArea.setText(preguntaActual.getTextoPregunta());
    textArea.setEditable(false);

    botonSi.setToggleGroup(group);
    botonNo.setToggleGroup(group);

}
```

*Ilustración 28: Método initialize de la clase ControladorPaginaTest. Elaboración propia*

Como podemos observar, se obtiene una referencia de la clase DAL para poder llevar a cabo operaciones sobre la base de datos, después, se asignan los manejadores de eventos a los botones “Continuar” y “Guardar y salir”. A continuación, se comprueba el atributo “tieneDPD” del objeto test, si es *true* (verdadero) quiere decir que el usuario ha respondido que su empresa si que tiene un delegado de protección de datos por lo que es necesario incluir las preguntas referentes al mismo en el cuestionario. Para hacer esto se usa el método “concatenarPreguntas” de la clase Test el cual ha sido explicado en el apartado dedicado a esta clase (página 22). Por último, se asignan valores a una serie de variables necesarias para el funcionamiento del cuestionario, estas son:

- Test: Referencia al objeto tipo Test.
- Respuestas: colección en la que serán almacenadas las respuestas que da el usuario.
- maximoPreguntas: Número de preguntas de las que consta el cuestionario.
- preguntaActual: Objeto de tipo pregunta que representa la pregunta que se muestra actualmente.
- numeroPregunta: El número de la pregunta actual
- textoNumeroPregunta: Texto que se sitúa en la parte superior de la ventana, está formado por el número de la pregunta actual más el número total de preguntas.

Después de asignar valore a los atributos, se asigna la variable “textoNumeroPregunta” al campo de texto de la parte superior de la ventana. El texto de la pregunta se asigna al área de texto que ocupa la zona central de la ventana, además se especifica que esta área de texto no puede ser modificada por el usuario. Por último, se agrupan los dos botones de respuesta lo cual nos facilitará más adelante la tarea de obtener la respuesta del usuario.

Una vez se ha ejecutado el método “initialize”, el usuario puede comenzar a responder el cuestionario, ya se ha cargado la primera pregunta si es un nuevo test o la pregunta por donde lo dejó si está continuando un test anterior. Cuando el usuario responda a una pregunta y pulse el botón para continuar a la siguiente se ejecuta el método “handleContinuar”.

```
public void handleContinuar(ActionEvent event){
    if(numeroPregunta != maximoPreguntas){
        actualizarPregunta();
    }else{
        if(getRespuesta()){
            finTest();
            mainApp.abrirVista("vistas/ResultadoTest.fxml");
        }
    }
}
```

*Ilustración 29: Método handleContinuar de la clase ControladorPaginaTest. Elaboración propia*

En este método, primero se comprueba si la pregunta actual es la última, de ser así se llama al método “finTest” el cual crea un nuevo informe con los resultados del test. Si no es la última pregunta se llama al método “actualizarPregunta” el cual utiliza el método “getRespuesta” para guardar la respuesta que ha dado el usuario, si no ha marcado ninguna opción de respuesta, se crea un mensaje en el cual se le avisa de que debe seleccionar una respuesta para poder continuar. La respuesta se guarda en la lista de respuestas del test para poder utilizarla posteriormente en la generación del informe.

Una vez se ha guardado la respuesta, el método “actualizarPregunta” crea una transición para que el cambio de pregunta sea fluido y agradable. El cambio consiste en incrementar el número de la pregunta, obtener la nueva pregunta de la lista de preguntas, cambiar el texto de la parte superior de la ventana en el cual se indica cual es el número de la pregunta actual, así como cuantas preguntas forman el cuestionario, por último, se cambia el texto del área de texto por el de la nueva cuestión.

```
public void actualizarPregunta(){

    if(getRespuesta()){
        try{
        FadeTransition fadeOut = new FadeTransition(Duration.millis(500),textArea);
        fadeOut.setAutoReverse(true);
        fadeOut.setFromValue(1.0);
        fadeOut.setToValue(0.0);

        fadeOut.setOnFinished(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                numeroPregunta = numeroPregunta + 1;
                preguntaActual = test.getPregunta(numeroPregunta - 1);
                textoNumeroPregunta = "Pregunta " + numeroPregunta +" de "+ maximoPreguntas;

                numeroPreguntaTextField.setText(textoNumeroPregunta);
                textArea.setText(preguntaActual.getTextoPregunta());
                FadeTransition fadeIn = new FadeTransition(Duration.millis(500),textArea);
                fadeIn.setAutoReverse(true);
                fadeIn.setFromValue(0.0);
                fadeIn.setToValue(1.0);
                fadeIn.play();
            }
        });
        fadeOut.play();

        }catch(Exception e){e.printStackTrace();}
    }
}
```

*Ilustración 30: Método actualizarPregunta de la clase ControladorPaginaTest. Elaboración propia*

Cuando termina un test, se genera un informe usando las respuestas dadas por el usuario, este proceso puede durar varios minutos por lo que para evitar que el usuario tenga la sensación de que la aplicación ha dejado de responder, mientras se genera el informe, se muestra una pequeña ventana con un icono de carga, así como con un mensaje en el que se informa al usuario de que el informe se está creando.


*Ilustración 31: Aviso dando cuenta de la generación del informe. Elaboración propia*

## 3.7 Persistencia de datos

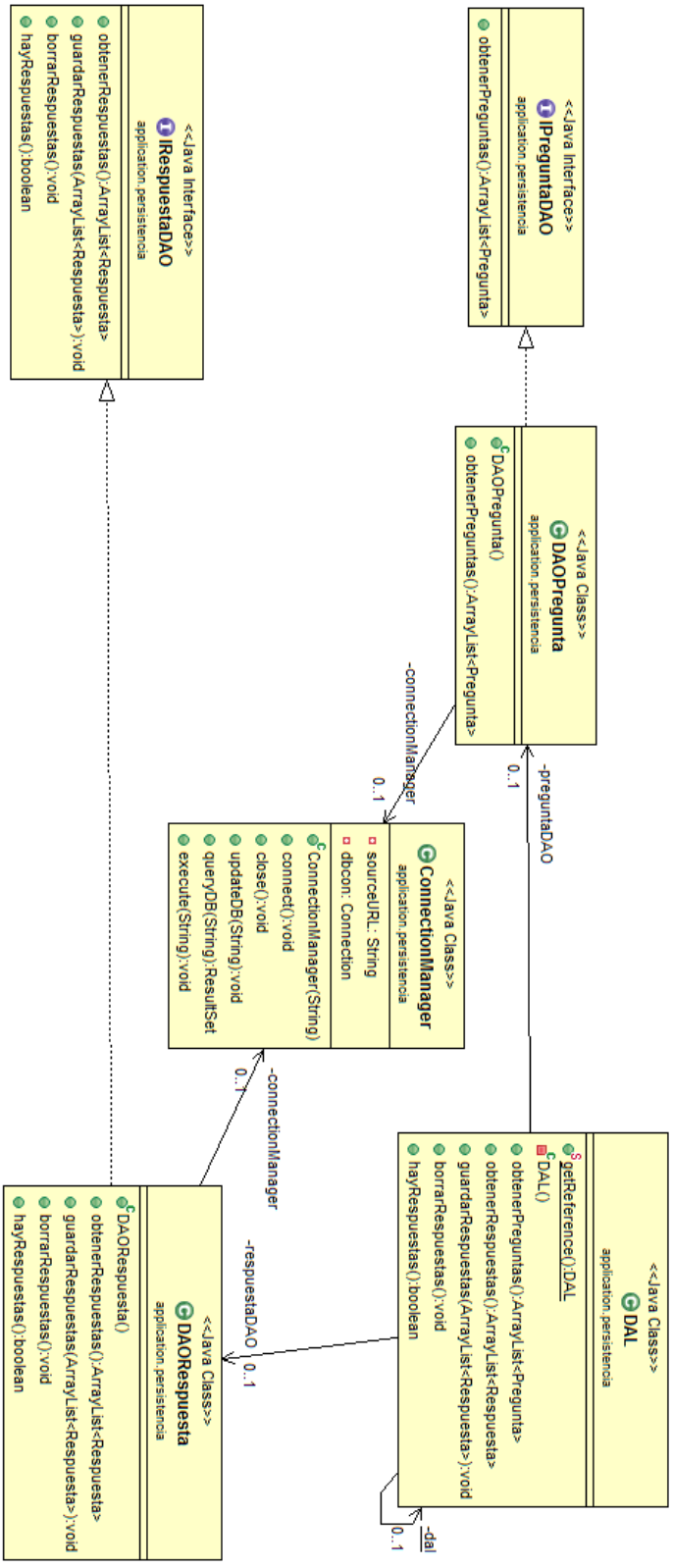


*Ilustración 32: Diagrama de clases con las clases de persistencia. Elaboración propia*

| Clase | Atributos | Métodos |
|---|---|---|
| **ConnectionManager** | sourceUrl<br>dbcon | connectionManager()<br>connect()<br>close()<br>update()<br>queryDB<br>execute() |
| **DAL** | Dal<br>preguntaDAO<br>respuestaDAO | getReference()<br>obtenerPreguntas()<br>obtenerRespuestas()<br>guardarRespustas()<br>borrarRespuestas()<br>hayRespuestas()<br>obtenerPreguntasDPD() |
| **DAOPregunta** | connectionManager | DAOPreguntas()<br>obtenerPreguntas()<br>obtenerPreguntasDPD() |
| **DAORespuesta** | connectionManager | DAORespuesta()<br>obtenerRespuestas()<br>guardarRespuestas()<br>borrarRespuestas()<br>hayRespuestas() |
| **IPreguntaDAO** | | obtenerPreguntas() |
| **IRespuestaDAO** | | obtenerRespuestas()<br>guardarRespuestas()<br>borrarRespuestas()<br>hayRespuestas() |

La base de datos empleada en la aplicación es una base de datos relacional formada por dos tablas. La tabla Preguntas en la cual se almacenan las preguntas que formarán el cuestionario y la tabla Respuestas que se utiliza para almacenar las respuestas que ha dado el usuario si este decide salir del test antes de terminarlo, de esta manera, posteriormente se puede continuar el cuestionario desde el punto en el que lo dejó.

La tabla Preguntas contiene el número de la pregunta el cual es la clave primaria, el texto de la pegunta, la respuesta que se considera correcta, el grupo al que pertenece la pregunta y el texto que aparecerá en el informe si el usuario no responde correctamente.

La tabla Respuestas solo consta de dos registros el número de la pregunta y la respuesta dada por el usuario a la misma.

Se ha utilizado la librería *SQLite* la cual implementa un motor de base de datos relacional *SQL [9]* integrado por lo que, a diferencia de otros motores, *SQLite* no necesita de un servidor por separado, además se trata de un motor rápido, transaccional y que no necesita de una configuración previa para utilizarlo.

A continuación, se va a detallar el diseño de la capa de persistencia de la aplicación, explicando que clases la forman y cuál es su funcionamiento.

Como se vio en el apartado de arquitectura, el paquete de persistencia está formado por la clase “ConnectionManager”, “DAL”, “DAOPregunta”, “DAORespuesta” y por las interfaces “IPreguntaDAO” e “IRespuestaDAO”.

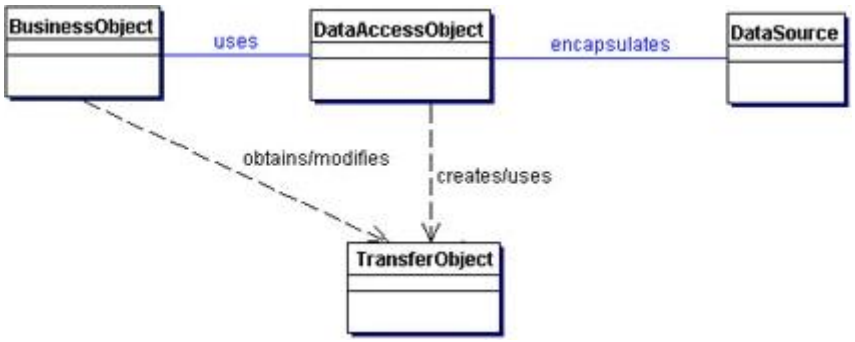


*Ilustración 33: Patrón DAO. Oracle.*
*http://www.oracle.com/ocom/groups/public/@otn/documents/digitalasset/146804.jpg*

En la capa de persistencia, se ha implementado el patrón *data acces object* (DAO), este patrón permite “abstraer y encapsular el acceso a la base de datos” (Oracle, 2002) [19] de manera que la aplicación no tiene en cuenta la procedencia de los datos que está utilizando, por lo tanto, la lógica de la aplicación funciona de manera independiente a la implementación de la base de datos, de esta manera se consigue dotar de mayor flexibilidad a la aplicación haciendo posible añadir diversos tipos de bases de datos o cambiar la base de datos relacional por otra distinta sin que se necesite modificar la lógica del programa.

Para implementar este patrón se han creado dos interfaces, “IPreguntaDAO” e “IRespuestaDAO” para las preguntas y las respuestas respectivamente. La clase “ConnectionManager” gestiona la conexión con la base de datos y contiene métodos para ejecutar las sentencias SQL. Las clases “PreguntaDAO” y “RespuestaDAO” son implementaciones de las interfaces mencionadas anteriormente, por último, la clase DAL proporciona métodos para que la lógica de la aplicación utilice la base de datos.

Esta aproximación hace que la aplicación sea mucho más mantenible y modificable en un futuro. Si se deseara añadir otro tipo de base de datos bastaría con realizar una nueva implementación de las interfaces.

A continuación, se van a explicar más en detalle estas clases.

La clase DAL es la encargada de comunicar la capa de persistencia con la capa lógica, cuando la aplicación necesita acceder a la base de datos, por ejemplo, para cargar las preguntas, utiliza una instancia de esta clase para hacerlo. En la clase DAL se ha implementado el patrón *Singleton* de forma idéntica a la clase Test.

```
public ArrayList<Respuesta> obtenerRespuestas(){
    try {
        return respuestaDAO.obtenerRespuestas();
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}

public void guardarRespuestas(ArrayList<Respuesta> respuestas){
    try {
        respuestaDAO.guardarRespuestas(respuestas);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public void borrarRespuestas(){
    try{
        respuestaDAO.borrarRespuestas();
    }catch(Exception e){
        e.printStackTrace();
    }
}
```

*Ilustración 34: Métodos obtenerRespuestas, guardarRespuestas y borrarRespuestas de la clase DAL. Elaboración propia.*

Como se puede observar, los métodos de la clase DAL utilizan las clases los métodos de las clases DAORespuesta y DAOPregunta. De esta manera, la clase DAL actúa como una interfaz entre la capa de persistencia y la capa de la lógica de la aplicación.

Como se ha explicado en el apartado 3.6, cuando se necesita acceder a la base de datos para realizar una consulta, se obtiene la referencia al DAL, posteriormente se realiza la consulta usando los métodos que proporciona la clase para tal efecto.

Al trabajar la capa de lógica siempre con el DAL, se consigue separar la implementación de la lógica de la implementación de la capa de persistencia, gracias a esta independencia, es posible realizar modificaciones en la capa de persistencia sin que se tenga que modificar de manera significativa ni el DAL ni la capa de la lógica, de esta manera se consigue incrementar la mantenibilidad de la aplicación.

Pero no solo la independencia entre capas aumenta la mantenibilidad, también lo hace la independencia entre los componentes de una capa, en la capa de persistencia se han utilizado las interfaces “IPreguntaDAO” e “IRespuestaDAO”.

```
public interface IPreguntaDAO {

    public ArrayList<Pregunta> obtenerPreguntas() throws Exception;

    public ArrayList<Pregunta> obtenerPreguntasDPD() throws Exception;

}
```

*Ilustración 35: Interfaz IPreguntaDAO. Elaboración propia*

```
public interface IRespuestaDAO {

    public ArrayList<Respuesta> obtenerRespuestas() throws Exception;

    public void guardarRespuestas(ArrayList<Respuesta> respuestas) throws Exception;

    public void borrarRespuestas() throws Exception;

    public boolean hayRespuestas() throws Exception;
}
```

*Ilustración 36: Interfaz IRespuestaDAO. Elaboración propia*

Las interfaces “IPreguntaDAO” e “IRespuestaDAO” tienen los métodos que deben implementar las clases “DAOPregunta” y “DAORespuesta”, como se ha comentado anteriormente, las interfaces permiten una mayor flexibilidad ya que delegan la implementación de sus métodos a las clases que las usen.

De esta manera es posible tener varias clases que usan una misma interfaz pero que la implementan de manera distinta. En nuestro caso esto puede servir para tener varias bases de datos diferentes por lo que tendríamos varias implementaciones de las interfaces, una para cada tipo de base de datos.

Vemos ahora el funcionamiento de las clase DAOPregunta y DAORespuesta. Estas clases son la parte fundamental del patrón DAO ya que son las encargadas de realizar las consultas a la base da datos.

La clase “DAOPregunta” implementa la interfaz “IPreguntaDAO”, la clase tiene un constructor en el cual se crea un objeto de tipo “ConnectionManager” especificando el nombre de la base de datos a la cual nos queremos conectar, en nuestro caso, nos conectamos a la base de datos llamada persistencia la cual es la única existente.

```
public DAOPregunta() throws Exception {
    super();
    try {
        connectionManager = new ConnectionManager("persistencia");
    } catch (ClassNotFoundException e) {
        throw new Exception(e);
    }
}
```

*Ilustración 37: Constructor de la clase DAOPregunta. Elaboración propia.*

```
@Override
public ArrayList<Pregunta> obtenerPreguntas() throws Exception {

    connectionManager.connect();
    System.out.println("Buscando preguntas en BBDD");
    ResultSet rs = connectionManager.queryDB("select * from Preguntas where "
            + "Grupo != 'Delegado de proteccion de datos' order by Grupo ");

    ArrayList<Pregunta> res = new ArrayList<Pregunta>();
    while(rs.next()){
        System.out.println("cargando...");
        res.add(new Pregunta(rs.getInt(1), rs.getString(2), "",
                rs.getString(5), rs.getString(4), rs.getString(3)));
    }
    connectionManager.close();
    return res;
}
```

*Ilustración 38: Método obtenerPreguntas de la clase DAOPreguntas. Elaboración propia*

El método “obtenerPreguntas” devuelve una colección con todas las preguntas existentes en la base de datos cuyo grupo no sea delegado de protección de datos, en primer lugar, crea una conexión con la base de datos usando el objeto “connectionManager” el cual se ha creado en el constructor. Una vez se ha establecido la conexión, se utiliza el método “queryDB” para realizar la consulta *SQL* a la base de datos, tras realizar la consulta se recorre el *result set* (conjunto de resultados) que nos ha devuelto la consulta. Para cada elemento del conjunto se crea un nuevo objeto de tipo pregunta, este objeto se añade a la colección de preguntas.

Una vez se han creado todas las preguntas, se cierra la conexión con la base de datos y se devuelve la colección de preguntas creadas que será empleada por la aplicación.

El método “obtenerPreguntasDPD” tiene la misma implementación que el anterior, pero devuelve únicamente aquellas preguntas cuyo grupo es “Delegado de protección de datos”.

En cuanto la clase “DAORespuestas”, implementa la interfaz “IRespuestaDAO”, se encarga de acceder a la base de datos para obtener las respuestas al cuestionario que haya almacenadas.

```java
public ArrayList<Respuesta> obtenerRespuestas() throws Exception {

    connectionManager.connect();
    System.out.println("Buscando preguntas en BBDD");

    ResultSet rs = connectionManager.queryDB("select * from Respuestas");
    ArrayList<Respuesta> res = new ArrayList<Respuesta>();

    while(rs.next()){
        String textoRespuesta = rs.getString(2);
        Respuesta respuesta = new Respuesta(rs.getInt(1), textoRespuesta);
        res.add(respuesta);
    }

    return res;
}

public void guardarRespuestas(ArrayList<Respuesta> respuestas) throws Exception{
    connectionManager.connect();
    System.out.println("Añadiendo respuestas a BBDD");
    String sql = "";
    Respuesta r;

    for(int i = 0; i< respuestas.size(); i++){
        r = respuestas.get(i);
        sql = "insert into Respuestas values("+ r.getNumeroPregunta()+","+"'"+r.getRespuesta()+"'"+")";
        System.out.println(sql);
        connectionManager.execute(sql);
    }
}
```

*Ilustración 39: Métodos obtenerRespuestas y guardarRespuestas de la clase DAORespuesta. Elaboración propia*

Tanto el constructor como el método “obtenerRespuestas” son idénticos al constructor y al método “obtenerPreguntas” visto anteriormente, la única diferencia es la consulta *SQL*, que en este caso accede a la tabla respuestas de la base de datos.

En cuanto al método “guardarRespuestas”, se le pasa como parámetro una lista de respuestas, se recorre esta lista y, para cada respuesta, esta se inserta en la base de datos.

```java
@Override
public void borrarRespuestas() throws Exception {
    // TODO Auto-generated method stub
    connectionManager.connect();
    System.out.println("Borrando registros de Respuestas");
    connectionManager.execute("delete from Respuestas");
}

@Override
public boolean hayRespuestas() throws Exception{

    connectionManager.connect();
    System.out.println("Borrando registros de Respuestas");
    ResultSet rs = connectionManager.queryDB("select count() from Respuestas");

    if(rs.getInt(1) != 0){
        return true;
    }else{
        return false;
    }
}
```

*Ilustración 40: Métodos borrarRespuestas y hayRespuestas de la clase DAORespuesta. Elaboración propia*

El método borrar respuestas ejecuta una sentencia SQL para borrar la tabla repuestas. En cuanto al método "hayRespuestas", realiza una consulta para saber cuántas respuestas hay en la base de datos, la consulta nos devuelve el número de entradas en la tabla respuestas tras lo cual se comprueba si el valor es cero en cuyo caso devolvemos true y en caso contrario false.

### 3.7.1 DB Browser for SQLite

Para editar de forma rápida el contenido de la base de datos, se puede utilizar la herramienta *DB Browser for SQLite*, este programa nos permite crear, diseñar y editar bases de dato compatibles con *SQLite*. Está diseñado para usuarios y desarrolladores que quieren crear bases de datos, buscar y editar datos de forma rápida y sencilla. No se necesitan conocimientos de *SQL* para utilizar esta herramienta (aunque es recomendable) lo que permite que sea empleada por cualquier tipo de usuario.

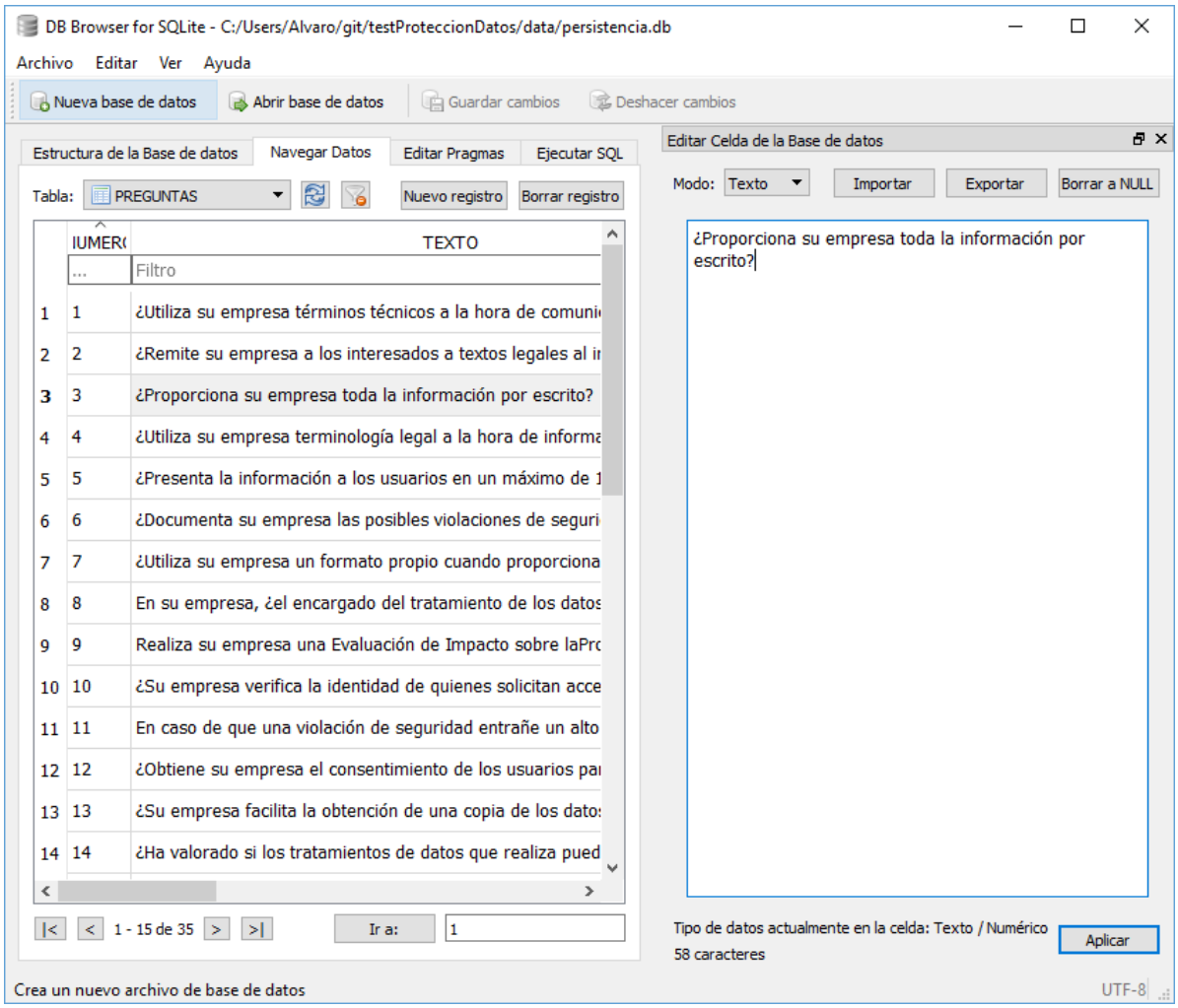


*Ilustración 41: DB Browser for SQLite. Elaboración propia.*

## 3.8 Apartado Gráfico: JavaFX

Como ya se ha comentado, para el apartado de la interfaz se ha empleado la librería JavaFX, esta es una librería gratuita creada por Oracle. Esta permite la creación de interfaces de forma sencilla.

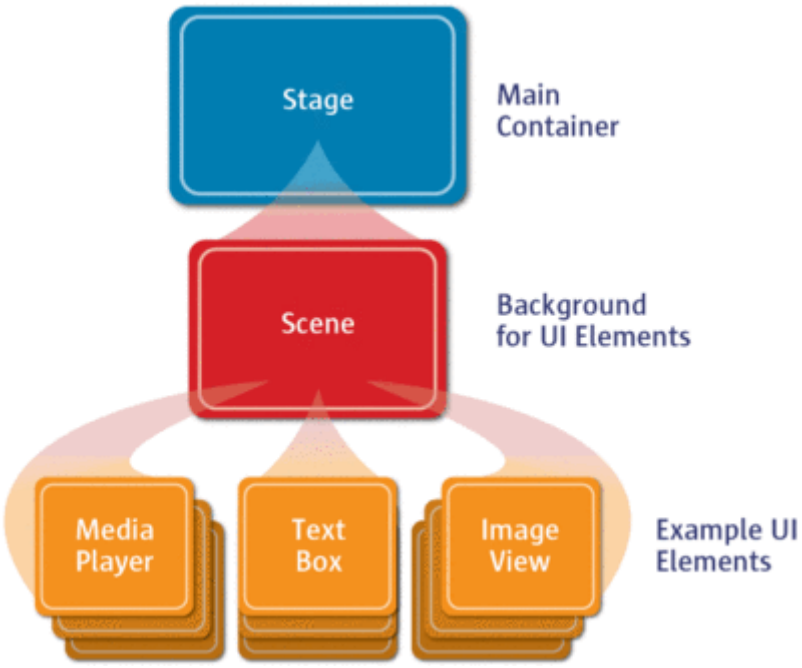


*Ilustración 42: Esquema básico de una aplicación JavaFX. Code Machery. http://code.makery.ch/library/javafx-8-tutorial/es/part1/*

Una interfaz realizada empleando la librería JavaFX consta de un único escenario (*stage*) el cual contendrá las escenas (*scenes*). Una aplicación puede tener múltiples escenas y en cada una de ellas está formada por un conjunto de elementos gráficos como botones, textos, paneles, etc. Una escena se define en un archivo *FXML* donde se indica que elementos la forman, su posición dentro de la escena y que controlador tendrá esa escena.

Un controlador es una clase java que se encarga de dotar de funcionalidad a la escena, siendo la encargada de responder a las acciones del usuario de la forma esperada como, por ejemplo, cerrando una ventana cuando el usuario pulse el botón establecido a tal efecto.

Para cambiar una escena por otra como ocurre cuando el usuario pulsa el botón de comenzar test y se produce la transición entre la ventana principal y la ventana en la que se muestran las instrucciones sobre el cuestionario, se crea una nueva escena usando el archivo *FXML* de la nueva ventana que queremos abrir y sustituimos la escena actual que se está mostrando en el escenario por la nueva escena. El cambio se produce con una transición de tipo difuminado en la que la ventana actual se difumina dejando paso a la nueva vista. De esta forma se consigue una transición entre vistas suave y elegante.

El proceso de pasar de una vista a otra se lleva a cabo en la clase “MainApp” con un método que tiene como parámetro el nombre del archivo *FXML* a partir del cual hay que crear la nueva escena. Los cambios de escenas se producen por lo general cuando el usuario realiza una determinada acción como pulsar un botón. En este caso, el controlador de la escena actual realizará la llamada al método pasando como parámetro el nombre del archivo que corresponda.

En esta aplicación el escenario se crea al iniciarse la misma en la clase “MainApp”. También se crea una escena a partir del archivo “VentanaPrincipal.fxml” y se le asigna como controlador la clase “ControladorVentanaPrincipal”.

La aplicación tiene en total de seis interfaces diferentes: la ventana principal, la de información sobre el test, la ventana en la que se pregunta al usuario sobre el delegado de protección de datos, la del test y la de resultados además de la ventana en la que se muestra el informe una vez ha concluido el cuestionario, pero esta, a diferencia del resto, se genera automáticamente sin usar la librería.

JavaFX se basa en el uso de archivos *FXML* los cuales es posible editar usando un editor de texto o usando la herramienta *scene builder* la cual nos permite crear las interfaces de una forma fácil y rápida. Esta aplicación nos permite seleccionar los elementos gráficos que queremos usar en nuestra ventana como botones, áreas de texto, tablas, etc.

La aplicación genera de forma automática los archivos *FXML* necesarios para poder utilizar las interfaces que hemos creado en nuestra aplicación.

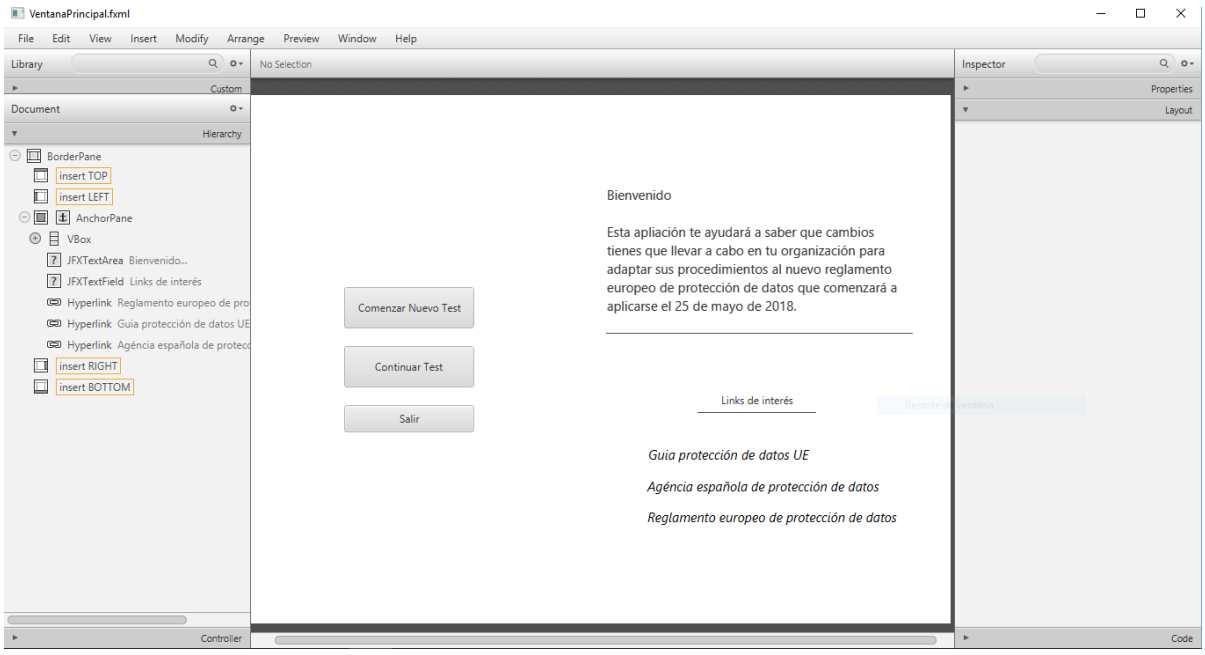


*Ilustración 43: SceneBuilder. Elaboración propia*

## 3.9 Generación de Informes

Una parte fundamental de la aplicación es la generación de un informe una vez terminado el test, es importante señalar que la aplicación solo se limita a realizar un informe de la situación de la empresa. Este informe debe servir para que un analista cualificado pueda recomendar las medidas a llevar a cabo a la vista de las deficiencias en materia de protección de datos que presenta la empresa en cuestión.

Para generar el informe se ha empleado la librería *Dynamic Reports* la cual permite crear informes de forma dinámica y sin la necesidad de un editor gráfico. Además, los informes creados pueden ser exportados a una gran cantidad de formatos como *PDF*, *DOCX* o XLS entre otros.

Cuando el usuario termina el cuestionario se crea un objeto de tipo *Dynamic Report* el cual creará el nuevo informe. Para crearlo, se divide el informe en tres categorías una para cada categoría de preguntas, a continuación, se recorre la colección de preguntas y se comprueba para cada pregunta si la respuesta dada por el usuario corresponde con aquella que se considera correcta, de no ser así, se añade al informe un texto indicando que indica que se ha contestado erróneamente la pregunta. Este texto se añade en la misma categoría a la que pertenece la pregunta.

En el caso de esta aplicación, en el informe aparecen todas aquellas preguntas que se han respondido de manera incorrecta (entendiendo por incorrecta aquellas que se han respondido de manera contraria a la que indica el RGPD) agrupadas en tres categorías: seguridad, delegado de protección de datos e información y derechos. De esta manera se obtiene un informe con todo aquello que la empresa debe corregir.

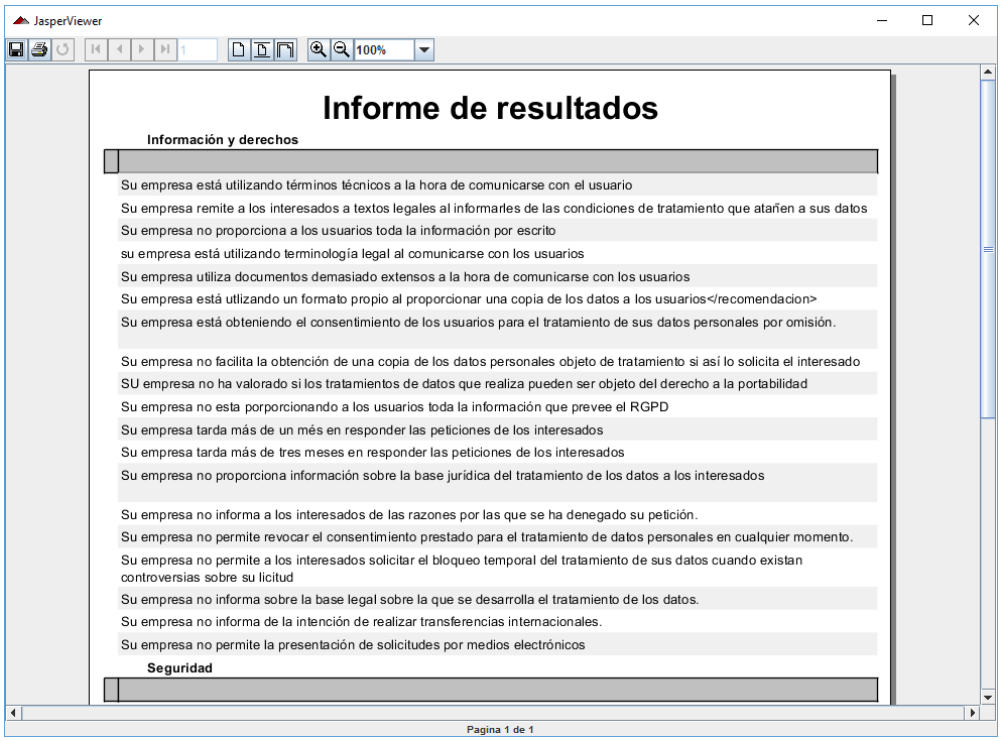


*Ilustración 44: Informe generado por la aplicación. Elaboración propia*

## 3.10 Pruebas Unitarias

Para la realización de las pruebas unitarias se ha decidido emplear la librería *JUnit* la cual nos permite crear de forma sencilla clases de prueba. Debido a que resulta extremadamente complejo automatizar las pruebas de la parte gráfica de la aplicación, se ha decidido centrar las pruebas automáticas en la lógica de la aplicación y la base de datos.

Para las pruebas de la base de datos, se ha utilizado una base de datos idéntica a la empleada por la aplicación sobre la que se han probado las diferentes operaciones que la aplicación puede llevar a cabo sobre la misma: recuperar las preguntas guardadas en la base de datos, recuperar las respuestas, guardar las respuestas, borrar las respuestas, comprobar si hay respuestas. Para usar esta nueva base de datos se han empleado versiones modificadas del DAL y de los DAOs.

Las pruebas unitarias se basan en el concepto de que cada prueba es independiente del resto por lo que el estado del programa debe ser el mismo al comenzar la prueba que al terminarla. En nuestro caso, ya que las pruebas se realizan sobre la base de datos, debemos asegurarnos de que cualquier cambio que realicemos a la misma durante una prueba debe ser borrado al finalizar esta. *JUnit* nos proporciona herramientas para conseguir esto, una de ellas es la posibilidad de designar uno o varios métodos para que se ejecuten después de cada prueba. En nuestro caso, solo modificamos la base de datos añadiendo o eliminando respuestas por lo que para garantizar que la base de datos no cambia entre una prueba y la siguiente, se ha implementado el método "resetBaseDatos" el cual borra cualquier respuesta que hallamos insertado.

El actuar de esta manera incrementa el tiempo que tardan las pruebas unitarias en ejecutarse, pero nos garantiza la independencia de las mismas.

A continuación, se va a explicar de forma detallada los métodos de prueba implementados.

```
@BeforeClass
public static void setUp(){
    try {
        dal = MockDAL.getSingleton();
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

@After
public void resetBaseDatos(){
    dal.borrarRespuestas();
}
```

*Ilustración 45: Método setUp de la clase TestBD. Elaboración propia*

El método “setUp” no es un método de pruebas propiamente dicho, pero sirve para inicializar la instancia del DAL que usarán el resto de métodos para realizar las consultas a la base de datos. Este método se ejecuta antes que cualquier otro método en la clase de pruebas gracias a la anotación “BeforeClass” que nos proporciona *JUnit*.

El método “resetBaseDatos” como ya se ha explicado anteriormente se ejecuta después de cada prueba y su función es borrar los posibles cambios realizados en la base de datos. Ya que la única tabla en la que se insertan datos es la de respuestas, este método elimina cualquier dato en esta tabla.

```
@Test
public void testRecuperarPreguntas() {
    ArrayList<Pregunta> preguntas = dal.obtenerPreguntas();
    Assert.assertEquals(preguntas.size(),15);
}
```

*Ilustración 46: Método testRecuperarPreguntas de la clase TestDB. Elaboración propia*

“testRecuperarPreguntas”: El objetivo de este test es comprobar que las preguntas se recuperan de forma correcta de la base de datos. En primer lugar, se realiza la consulta a la base de datos y a continuación se comprueba que el número de objetos recuperados coincide con el número de preguntas almacenadas en la base de datos.

```
@Test
public void testRecuperarRespuestas() {
    ArrayList<Respuesta> respuestas = new ArrayList<Respuesta>();
    respuestas.add(new Respuesta(1,"si"));
    dal.guardarRespuestas(respuestas);

    ArrayList<String> respuestasRecuperadas = dal.obtenerRespuestas();
    Assert.assertEquals(respuestasRecuperadas.size(),1);
}
```

*Ilustración 47: Método testRecuperarRespuestas de la clase TestDB. Elaboración propia*

“testRecuperarRespuestas”: El objetivo de este test es comprobar que las respuestas se recuperan de manera correcta de la base de datos. En primer lugar, se inserta en la base de datos una respuesta, a continuación se realiza la consulta y se comprueba que se ha recuperado un objeto y que este coincide con el objeto que se había insertado previamente.

```
@Test
public void testBorrarRespuestas() {
    ArrayList<Respuesta> respuestas = new ArrayList<Respuesta>();
    respuestas.add(new Respuesta(1,"si"));
    dal.guardarRespuestas(respuestas);

    dal.borrarRespuestas();
    Assert.assertFalse(dal.hayRespuestas());
}
```

*Ilustración 48: Método testBorrarRespuestas de la clase TestDB. Elaboración propia*

"testBorrarRespuestas": El objetivo de este test es comprobar que las respuestas se borran correctamente de la base de datos. De nuevo, se inserta una respuesta en la base de datos para posteriormente borrarla y comprobar que ya no hay ninguna respuesta almacenada.

```
@Test
public void testNoHayRespuestas(){
    boolean res = dal.hayRespuestas();
    Assert.assertEquals(res,false);
}

@Test
public void testHayRespuestas(){

    ArrayList<Respuesta> respuestas = new ArrayList<Respuesta>();
    respuestas.add(new Respuesta(1,"si"));
    dal.guardarRespuestas(respuestas);

    boolean res = dal.hayRespuestas();
    Assert.assertEquals(res,true);
}
```

*Ilustración 49: Métodos testNoHayRespuesta y testHayRespuesta de la clase TestDB. Elaboración propia*

"testHayRespuestas" y "testNoHayRespuestas":

El objetivo de estos test es comprobar que le método que indica si hay o no respuestas almacenadas funciona correctamente. Para el primer test, Se inserta una respuesta en la base de datos y se ejecuta el método, a continuación, se comprueba que ha devuelto como resultado *true*. En cuanto al segundo, no se inserta ninguna respuesta. Al ejecutar el método se comprueba que el resultado es *false*.

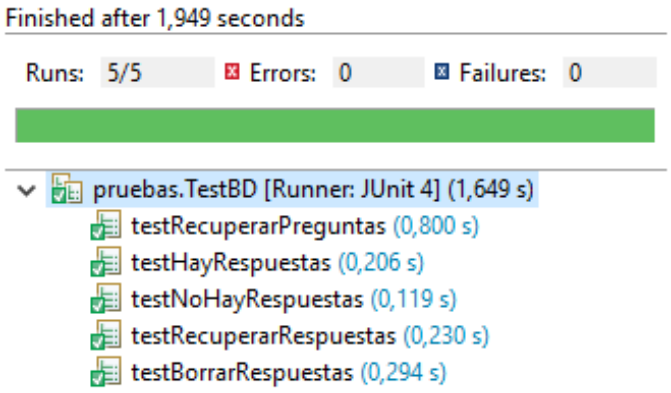


*Ilustración 50: Resultado de la ejecución de las pruebas. Elaboración propia.*

# 4 Guía de Uso

Una vez inicializada la aplicación, se le muestra al usuario la ventana principal (Ilustración 21). En ella aparece una breve descripción de la aplicación, así como un mensaje de bienvenida y unos enlaces a documentos y sitios de interés relacionados con el nuevo reglamento europeo de protección de datos.

Desde la ventana principal el usuario puede decidir si comienza el test, continua uno previamente guardado (en caso de haberlo) o cierra en la aplicación, si elige esta última opción se le preguntará si está seguro de querer salir. Si decide continuar el último test, dará comienzo el cuestionario por la pregunta por la que se quedó cuando salió, si por el contrario elije comenzar el test, la ventana cambiará y se le mostrará al usuario el siguiente mensaje:

"Va a comenzar el test para comprobar que cambios debe realizar su empresa para adaptarse al nuevo reglamento europeo de protección de datos. El test consta de 35 preguntas. Para cada pregunta seleccione la respuesta que considera adecuada, al finalizar el test, se generará un informe recopilando las deficiencias detectadas, puede usar este informe para llevar a cabo las reformas necesarias con el fin de subsanar dichas deficiencias. En cualquier momento puede usar el botón de volver para guardar el progreso realizado y salir del test."

Tras pulsar el botón comenzar, se pregunta al usuario si en su empresa existe la figura del delegado de protección de datos (Ilustración 18). Dependiendo de la respuesta que dé el usuario, el cuestionario se configura con preguntas sobre el delegado de protección de datos en caso de que la respuesta sea afirmativa o sin ellas en caso contrario.

Una vez contestada esta pregunta, si el usuario pulsa el botón "Continuar" se le muestra la primera de las preguntas que forman el cuestionario, en la parte superior de la pantalla se informa del número de pregunta actual, así como del número total de preguntas que tendrá que responder el usuario. A continuación, se muestra el texto de la pregunta seguido de las posibles respuestas (Ilustración 27).

El usuario debe elegir una de las respuestas y pulsar el botón de continuar. Si no ha seleccionado ninguna respuesta se le mostrará un mensaje encomiándole a hacerlo, de igual modo si pulsa el botón "Guardar y salir", se le mostrará un mensaje donde se le informa de que las respuestas que ha dado serán almacenadas para que pueda proseguir con el cuestionario cuando lo desee y pidiéndole que confirme si realmente desea abandonarlo.

Una vez se ha finalizado el test, comienza la generación del informe de forma automática, para proporcionar información al usuario se muestra una pequeña ventana con el texto "Generando informe" y un icono de carga, de esta manera el usuario sabe que la aplicación está funcionando correctamente y que tiene que esperar a que se complete la generación del informe (Ilustración 31).

Cuando finaliza la creación del informe se abre una ventana en la que es posible ver el informe generado y desde la que se puede exportar el informe a diferentes formatos como *PDF*, *XLS* o *DOCX*, así como imprimirlo o enviarlo por correo electrónico.

Una vez el usuario ha terminado con el informe y decide cerrar la ventana, la aplicación le da la opción de volver a la pantalla principal o volver a generar el informe en caso de que así lo desee. Es importante señalar que una vez que el usuario vuelve a la ventana de inicio, el informe generado se pierde siendo necesario volver a realizar el test en caso de querer generar uno nuevo. La aplicación informa al usuario de este hecho en el momento en el que pulsa el botón para regresar al inicio dándosele la opción de permanecer en la venta actual o de continuar.

Tras exportar el informe al formato deseado este deberá ser analizado por un especialista el cual debe valorar y recomendar las acciones que debe llevar a cabo la empresa. Esta labor puede ser llevada a cabo por personal de la empresa con la correcta formación, pero dado que la aplicación va dirigida a empresas pequeñas que pueden carecer de dichos recursos, es recomendable que sea un profesional el encargado de valorar el informe.

# 5 Conclusiones

En este documento hemos podido comprobar la problemática de la adaptación de las empresas al nuevo reglamento europeo de protección de datos. Se ha conseguido desarrollar una aplicación que cumple con los objetivos establecidos al comienzo de este proyecto y que consigue solucionar la problemática inicial, ayudando a las pequeñas empresas a realizar una evaluación de sus procedimientos de manera rápida y cómoda. Esto es posible gracias a la flexibilidad que nos proporciona el cuestionario al que se pueden añadir, eliminar o modificar preguntas de forma sencilla gracias a la herramienta "*db browser for sqlite*.

La aplicación desarrollada es una aplicación robusta, muy fácil de usar y accesible para los nuevos usuarios. Además, se trata de una aplicación muy mantenibe, que gracias a su diseño modular hace que sea muy sencillo realizar modificaciones a la misma.

## Posibles mejoras futuras

Como posibles mejoras futuras se podría implementar un sistema de cuentas de usuario de manera que la aplicación pudiera ser utilizada por varias personas y que estas tuvieran cada una un perfil de usuario desde el que poder acceder a sus test guardados, así como a los informes generados. También se podría intentar integrar el gestor de bases de datos en la aplicación o incluso realizar otra aplicación que sirviera a los administradores para gestionar tanto la base de datos como los informes generados.

También se podría incluir una función para modificar el idioma de la aplicación y hacerla accesible a muchos más usuarios de esta manera.

## Valoración personal

La realización de este proyecto me ha dado la posibilidad de investigar y aprender más sobre la legislación de protección de datos, su importancia y su aplicación en la industria. En cuanto al desarrollo del proyecto, este ha sido sencillo en gran medida gracias a que las tecnologías empleadas (Java, JavaFX, Eclipse) las he podido emplear en el transcurso de mis estudios en el grado de ingeniería informática. Únicamente el uso de tecnología de generación automática de informes (*Dynamic Reports*) ha presentado algún problema, debido a mi desconocimiento sobre ella, pero haber empleado esta tecnología me ha permitido aprender nuevas habilidades que estoy seguro me serán de utilidad en mi carrera profesional.

Estoy satisfecho con el resultado final ya que creo que se ha conseguido desarrollar una aplicación interesante y de utilidad, que cumple con los requisitos establecidos.

# 6 Anexo

## Listado de preguntas

### Información y derechos

- ¿Utiliza su empresa términos técnicos a la hora de comunicarse con los usuarios?
- ¿Remite su empresa a los interesados a textos legales al informarles de las condiciones de tratamiento que atañen a sus datos?
- ¿Proporciona su empresa toda la información por escrito?
- ¿Utiliza su empresa terminología legal a la hora de informar a los interesados?
- ¿Presenta la información a los usuarios en un máximo de 15 líneas?
- ¿Documenta su empresa las posibles violaciones de seguridad?
- ¿Utiliza su empresa un formato propio cuando proporciona una copia de los datos a los interesados?
- ¿Obtiene su empresa el consentimiento de los usuarios para el tratamiento de sus datos personales por omisión?
- ¿Su empresa facilita la obtención de una copia de los datos personales objeto de tratamiento si así lo solicita el interesado?
- ¿Ha valorado si los tratamientos de datos que realiza pueden ser objeto del derecho a la portabilidad?
- ¿La información que proporciona a los interesados contiene todos los elementos que prevé el RGPD como la base jurídica del tratamiento, la intención de realizar transferencias internacionales, los datos del Delegado de Protección de Datos (si lo hubiera) ...?
- ¿Responde su empresa en menos de un mes a las peticiones de los interesados?
- ¿Su empresa tarda más de tres meses en responder a peticiones de los interesados?
- ¿Proporciona su empresa información sobre la base jurídica del tratamiento de los datos a los interesados?

- En caso de que su empresa decida no atender una solicitud, ¿informa de ello al interesado, motivando su negativa?

- ¿Permite su empresa revocar el consentimiento prestado para el tratamiento de datos personales en cualquier momento?

- ¿Permite su empresa a los interesados solicitar el bloqueo temporal del tratamiento de sus datos cuando existan controversias sobre su solicitud?

- En el momento de recoger los datos de los interesados, ¿Informa su empresa sobre la base legal sobre la que se desarrolla el tratamiento de los datos?

- ¿Informa su empresa de la Intención de realizar transferencias de datos internacionales?

- ¿Su empresa permite la presentación de solicitudes por medios electrónicos?

**Seguridad**

- ¿Documenta su empresa las posibles violaciones de seguridad?

- En su empresa, ¿el encargado del tratamiento de los datos mantiene un registro de las actividades de tratamiento realizadas a dichos datos?

- ¿Realiza su empresa una Evaluación de Impacto sobre la Protección de Datos (EIPD) antes de la puesta en marcha de tratamientos que puedan suponer un riesgo para los derechos de los interesados?

- ¿Su empresa verifica la identidad de quienes solicitan acceso y de quienes ejerzan los restantes derechos ARCO?

- ¿Dispone su empresa de un registro o herramienta similar en que pueda documentar los incidentes de seguridad que se produzcan, aunque no sean notificados a las autoridades de protección de datos?

- ¿Notifica su empresa las violaciones de seguridad en un plazo de setenta y dos horas como máximo?

- ¿Realiza su empresa una valoración del riesgo de los tratamientos de datos que realiza?

- ¿Dispone su empresa de una metodología para la realización de la Evaluación de Impacto?

**Delegado de protección de datos**

- ¿Son públicos los datos de contacto de su delegado de protección de datos?
- ¿Cuenta el delegado de protección de datos con total autonomía dentro de su empresa?
- ¿Ha establecido su empresa procedimientos para que los interesados contacten con el delegado de protección de datos?
- ¿Comunica su empresa la designación del delegado de protección de datos, así como sus datos de contacto a las autoridades de supervisión?
- ¿Ha establecido los criterios para para valorar las cualificaciones profesionales y los conocimientos a la hora de seleccionar al Delegado de Protección de Datos?
- ¿Responde el delegado de protección de datos ante el nivel superior de la dirección de la empresa?

# 7 Glosario de Términos

[1] *Handler:* Código asociado a algún tipo de evento (pulsación de una tecla o botón) el cual se ejecuta al producirse el evento al que está asociado.

[2] Librería: También llamada *framework,* es una colección de subprogramas cuya función es proporcionar servicios a otro software.

[3] Referencia: Es una variable que permite al programa acceder a datos guardados en memoria.

[4] Parámetro: Variable utilizada para recibir valores de entrada en un método.

[5] FXML: Lenguaje basado en XML que proporciona la estructura necesaria para crear interfaces independientes de la lógica de la aplicación.

[6] *Data source*: Contenedor estructurado de datos que se emplea para poblar de datos un informe.

[7] Punto de entrada: Procedimiento de inicio de un programa.

[8] Hojas de Estilo en Cascada (*Cascading Style Sheets*): Es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado. Es muy usado para establecer el diseño visual de páginas web e interfaces de usuario.

*[9] SQL (Structured Query Language):* Lenguaje específico del dominio que permite manipular datos en bases de datos relacionales.

# 8 Bibliografía

[1] Padín Vidal, Alejandro. 2016. *Nuevo Reglamento General de Protección de Datos de la Unión Europea. Counsel en Garrigues.*

[2] A un click de las TIC, *"Nuevo Reglamento europeo de protección de datos y su adaptación a la era digital"*. http://aunclicdelastic.blogthinkbig.com/nuevo-reglamento-europeo-de-proteccion-de-datos/ . 29 de feb. 2016.

[3] Hay Derecho. "*Novedades e implicaciones prácticas del nuevo Reglamento General de Protección de Datos Personales de la UE*". http:/hayderecho.com/2017/02/15/novedades-e-implicaciones-practicas-del-nuevo-reglamento-general-de-proteccion-de-datos-personales-de-la-ue/. 15 feb. 2017.

[4] NetApp (2017), *"Una encuesta realizada entre responsables tecnológicos europeos desvela que la adecuación al RGPD sigue resultando un proceso lento y confuso"*. < http://www.netapp.com/es/company/news/press-releases/news-rel-20170508-845692.aspx>

[5] Prieto Sáez, Natividad, [et al]. 2013."Objetos, clases y programas" en "Aprender a programar usando Java". Valencia: Editorial Universitat Politècnica de València. Página 15.

[6] Wikipedia. "Software development process". https://en.wikipedia.org/wiki/Software_development_process#Waterfall_development. 30 jun. 2017.

[7] Wikipedia. "Singleton". https://es.wikipedia.org/wiki/Singleton. 23 agosto 2017.

[8] Agencia Española de Protección de Datos. *"Reglamento General de Protección de Datos"*. https://www.agpd.es/portalwebAGPD/temas/reglamento/index-ides-idphp.php. 10 agosto 2017.

[9] *"Guia de reglamento general de protección de datos para responsables del tratamiento"*. Agencia española de protección de datos. https://www.agpd.es/portalwebAGPD/temas/reglamento/common/pdf/guia_rgpd.pdf"

[10] Wikipedia. "Modelo–vista–controlador". https://es.wikipedia.org/wiki/Modelo-vista-controlador. 25 julio 2017.

[11] *"Reglamento general de protección de datos". Agencia catalana de protección de datos*. http://apdcat.gencat.cat/es/documentacio/RGPD/. 17 junio 2017

[12] *"REGLAMENTO (UE) 2016/679 DEL PARLAMENTO EUROPEO Y DEL CONSEJO de 27 de abril de 2016 relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos y por el que se deroga la Directiva 95/46/CE (Reglamento general de protección de datos)".* 27 de abril de 2016. Parlamento Europeo.

[13] "Preparing for the General Data Protection Regulation (GDPR): 12 steps to take now". Information Comissioner Office. 25 mayo 2017. https://ico.org.uk/media/for-organisations/documents/1624219/preparing-for-the-gdpr-12-steps.pdf

[14] "Java Platform, Standard Edition (Java SE) 8". Oracle. http://docs.oracle.com/javase/8/javase-clienttechnologies.htm

[15] Wikipedia. "SQL", https://en.wikipedia.org/wiki/SQL. 28 agosto 2017.

[16] Wikipedia. “Eclipse (Software)”. https://es.wikipedia.org/wiki/Eclipse_(software). 31 agosto 2017.

[17] Bautista Perales, Ismael. (2014). “Aplicación Web de bases de datos usando el Framework Ruby on Rails”. http://hdl.handle.net/10251/38942. Universidad Politécnica de Valencia.

[18] Jasper Reports. *“JasperReports - Data Source Sample (version* 6.4.1)”. http://jasperreports.sourceforge.net/sample.reference/datasource/. 22 mayo 2017.

[19] Oracle. “Core J2EE Patterns - Data Access Object”. http://www.oracle.com/technetwork/java/dataaccessobject-138824.html. 2001.

[20]Wikipedia. “Punto de entrada (informática)”. https://es.wikipedia.org/wiki/Puto_de_entrada_(informática). 25 agosto 2017.